ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80*

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 5

ENCYCLOPEDIA for the TRS-80*

VOLUME 5

wayne GREEN inc
PETERBOROUGH NH 03458

*Trademark of Tandy Corp

ENCYCLOPEDIA for the TRS-80*

FIRST EDITION
FIRST PRINTING JANUARY 1982

Edited by Kate Comiskey and Katherine Putnam
Production Manager Margaret Baker
Technical Assistance by Jake Commander
and Kenniston W. Lord Jr., CDP
Illustrations by Howard Happ
Typeset by Karen Stewart

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

contents

BUSINESS

Hi Ho Silver!
Accountant's Aid

Hi Ho Silver!

by James J. Conroy

Our turbulent economy causes speculators around the world to invest vast sums in gems, gold, and silver. This has resulted in a phenomenal rise (and sometimes fall) in the prices of these commodities. How does a supplier of precious metals keep his or her extensive stock priced to maintain a profit margin? One solution is to use a microcomputer to calculate and print up-to-date price sheets.

Worth It's Weight in Silver

When the price of silver hit an astounding 50 dollars an ounce, my employer, who supplies silver stock to hobbyists, craftsmen, and silversmiths, was well prepared. While other jewelers and supply houses hesitated to quote prices on items and even refused to sell their silver stock, we were able to compute and print daily price sheets for our inventory by using our TRS-80 microcomputer.

Before the price boom in silver hit, I had been shown a fragment of a BASIC program which computed the cost of measured sheets of silver by multiplying the weight of each sheet (in troy ounces) by the cost per ounce of silver. (See Figure 1.) The program core did accomplish its intended purpose. (See Program Listing 1.) It read a data statement of weights and multiplied them by a dollar value which was input by the operator. It lacked formatted output, however, and it became my task to finish the job. Not wishing to redesign the program, I added the input and output formatting needed to prompt the operator to input information and have the computer print out the sheet size of the silver, its gauge (thickness), and its computed cost. Then a small problem arose.

More and More

When it was clear that the program was effective, the suggestion was made to include our silver wire stock in the computations. We decided that what we really needed was a comprehensive price sheet to give to customers. To accomplish this, I added more data statements, and soon we were calculating the prices of 53 different categories of silver. I found that by creating module routines (lines 180–470, 610–660, 710–770, and 820–880) to handle each category of silver, I had dispersed my data statements within the program. Because of this, I could read the data statements as they were called for. This eliminated the need for any RESTORE statements. You, however, might elect to list all data at the end of the program and read it into an array to be accessed when needed.

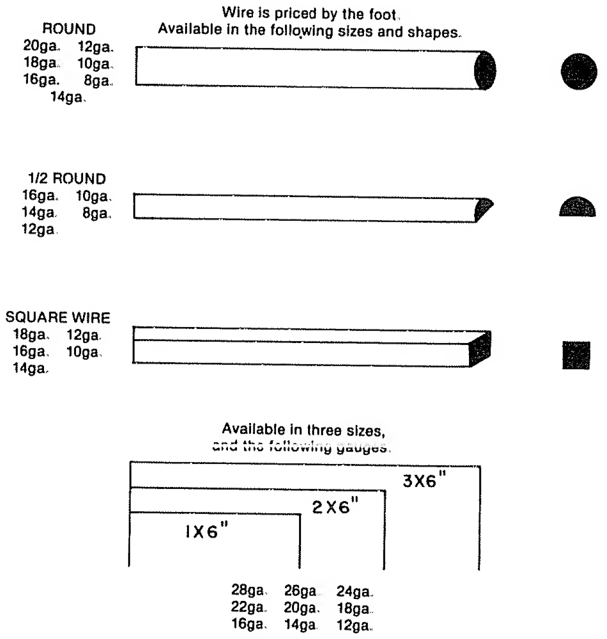


Figure 1. *A section from a page in our catalog. Sterling silver stock is manufactured to standard weights, gauges (thickness), and purity. The purity is called "fineness." 99% pure is 1000 fine. Sterling silver is rated 925 (or 92.5% pure silver). Silver is measured in troy ounces.*

GILMAN'S LAPIDARY SUPPLY—CURRENT SILVER PRICE LIST

DATE: 2—1—1980 CURRENT COST OF SILVER: $50.00 PER OUNCE

SHEET SILVER

Price	Dimension	Gauge
$20.40	1×6	28
$25.56	1×6	26
$31.98	1×6	24
$41.58	1×6	22
$51.75	1×6	20
$66.99	1×6	18
$83.07	1×6	16
$104.46	1×6	14
$129.75	1×6	12
$40.80	2×6	28
$51.12	2×6	26
$63.96	2×6	24
$83.16	2×6	22

$103.50	2 × 6	20
$133.99	2 × 6	18
$166.14	2 × 6	16
$208.92	2 × 6	14
$259.50	2 × 6	12
$61.20	3 × 6	28
$76.68	3 × 6	26
$95.94	3 × 6	24
$124.74	3 × 6	22
$155.25	3 × 6	20
$200.97	3 × 6	18
$249.21	3 × 6	16
$313.38	3 × 6	14
$389.25	3 × 6	12

ROUND WIRE

Price per Foot	Gauge
$107.00	4
$67.50	6
$42.60	8
$26.80	10
$16.85	12
$10.60	14
$6.65	16
$4.20	18
$2.65	20
$1.65	22
$1.05	24

HALF ROUND WIRE

Price per Foot	Gauge
$82.50	2
$47.00	4
$34.00	6
$21.20	8
$14.00	10
$8.40	12
$5.30	14
$3.05	16

SQUARE WIRE

Price per Foot	Gauge
$87.50	6
$54.50	8
$34.10	10

Example continued

$21.45	12
$13.45	14
$8.45	16
$5.35	18

PRICES SUBJECT TO CHANGE WITHOUT NOTICE

Example 1. *Sample price sheet*

This program allows you to modify your prices as quickly as the world market fluctuates. In keeping with normal, prudent business practices, we try to maintain some stability in our selling prices. But, if the precarious metal market shifts, we are ready.

This program's concept isn't limited to silver. We have used a similar concept to compute the retail and wholesale prices of some of our jewelry findings. The same theory applies to evaluating any stock item in gold, platinum, or other precious metal. For the investor, craftsman, or jeweler, the microcomputer could be a silent watchdog ensuring an adequate return on investments.

Program Listing 1. *The original program*

Encyclopedia Loader

```
10 INPUT P
20 FOR A = 1 TO 9:
    READ B(A):
    NEXT A
30 FOR A = 1 TO 9:
    LET C(A) = B(A) * 6:
    NEXT A
35 J = 30
40 FOR A = 1 TO 9
45  J = J - 2
50  G(A) = C(A) * P
55  LPRINT J;G(A)
60  NEXT A
70 DATA .0680,.0852,.1066,.1386,.1725,.2233,.2769,.3482,.4325
80 END
```

Program Listing 2. *Revised silver price sheet program*

```
 10 :
    ' SILVER PRICE SHEET
 20 :
    ' BY JAMES J. CONROY
 30 :
    ' 7 EAST GARRISON STREET
 40 :
    ' BETHLEHEM  PA  L80L8
 50 CLS
 60 PRINT "INPUT THE CURRENT PRICE OF SILVER (PER OZ.)":
    INPUT P
 70 PRINT "INPUT DATE (M,D,Y";:
    INPUT M:
    INPUT D:
    INPUT Y:
 80 PRINT "HOW MANY COPIES DO YOU WANT?";:
    INPUT Z
 90 FOR V = 1 TO Z
100  LPRINT TAB(6)"-GILMANS LAPIDARY SUPPLY -- CURRENT SILVER PRICE
     LIST -"
110  LPRINT "-----------------------------------------------------
     ------------"
120  LPRINT :
     LPRINT :
     LPRINT
130  LPRINT "DATE:";M;"-";D;"-";Y;:
     LPRINT "        CURRENT COST OF SILVER:";:
     LPRINT USING "$$##.##";P;:
     LPRINT " PER OUNCE"
140  LPRINT :
     LPRINT :
     LPRINT
150  IF V > 1
      THEN
       GOTO 250
160  :
     ' THIS READS WEIGHTS (ONE SQUARE INCH OF STERLING SILVER)
170  :
     ' AND ASSIGNS VALUES FOR EACH SHEET SIZE
180  FOR A = 1 TO 9 :
      READ B(A):
      NEXT A
190  FOR A = 1 TO 9 :
      LET C(A) = B(A) * 6:
```

Program continued

```
      NEXT A
200  FOR A = 1 TO 9 :
      LET X(A) = B(A) * 12:
      NEXT A
210  FOR A = 1 TO 9 :
      LET D(A) = B(A) * 18:
      NEXT A
220  FOR A = 1 TO 9 :
      LET E(A) = X(A) * P:
      NEXT A
230  FOR A = 1 TO 9 :
      LET F(A) = C(A) * P:
      NEXT A
240  FOR A = 1 TO 9 :
      LET G(A) = D(A) * P:
      NEXT A
250  GOSUB 530
260  H = 1:
     I = 6:
     J = 30
270  FOR A = 1 TO 9
280   J = J - 2
290   F(A) = INT(F(A) * 100 + .5) / 100
300   REM  TEN SPACES BETWEEN DIMENSION AND GAUGE
310   LPRINT USING "$$##.##";F(A);:
      LPRINT TAB(16)H;"X";I;"          ";J
320   NEXT A
330  GOSUB 540
340  J = 30:
     H = 2:
     I = 6
350  FOR A = 1 TO 9
360   J = J - 2
370   E(A) = INT(E(A) * 100 + .5) / 100
380   LPRINT USING "$$##.##";E(A);:
      LPRINT TAB(16)H;"X";I;"          ";J
390   NEXT A
400  GOSUB 540
410  J = 30:
     H = 3:
     I = 6
420  FOR A = 1 TO 9
430   J = J - 2
440   G(A) = INT(G(A) * 100 + .5) / 100
450   LPRINT USING "$$##.##";G(A);:
      LPRINT TAB(16) H;"X";I;"          ";J
460   NEXT A
470  DATA .0680,.0852,.1006,.1396,.1725,.2233,.2769,.3482,.4325
480  LPRINT
490  LPRINT ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     :::::::::::::::"
500  LPRINT CHR$(12)
510  LPRINT :
     LPRINT
520  GOTO 560
530  LPRINT "   COST"; TAB(16) "DIMENSION    GAUGE"
540  LPRINT "- - - - - - - - - - - - - - - - - - - - - - - - - - - -
      - - - - - - - -"
550  RETURN
560  LPRINT :
     LPRINT
570  LPRINT "ROUND WIRE"
580  LPRINT "----------"
590  LPRINT :
     LPRINT :
     LPRINT "PRICE PER FOOT             GAUGE":
     LPRINT :
     IF V > 1
      THEN
       GOTO 610
```

```
600  FOR W = 0 TO 10:
      READ R(W):
      NEXT W
610  G = 2
620  FOR A = 0 TO 10
630   C = P * R(A):
      G = G + 2
640   LPRINT USING "$$##.##";C;:
      LPRINT TAB(25) G
650   NEXT A
660  DATA 2.14,1.35,.825,.536,.337,.212,.133,.084,.053,.033,.021
670  LPRINT :
     LPRINT :
     LPRINT "HALF ROUND WIRE"
680  LPRINT "----------------":
     LPRINT :
     LPRINT
690  LPRINT "PRICE PER FOOT                GAUGE":
     LPRINT
700  IF V > 1
      THEN
       GOTO 720
710  FOR W = 0 TO 7:
      READ H(W):
      NEXT W
720  G = 0
730  FOR A = 0 TO 7
740   C = P * H(A):
      G = G + 2
750   LPRINT USING "$$##.##";C;:
      LPRINT TAB(25) G
760   NEXT A
770  DATA 1.65,.940,.680,.424,.280,.168,.106,.061
780  LPRINT :
     LPRINT :
     LPRINT "SQUARE WIRE":
     LPRINT "-----------"
790  LPRINT :
     LPRINT
800  LPRINT "PRICE PER FOOT                GAUGE":
     LPRINT
810  IF V > 1
      THEN
       GOTO 830
820  FOR W = 0 TO 6:
      READ S(W):
      NEXT W
830  G = 4
840  FOR A = 0 TO 6
850   C = P * S(A):
      G = G + 2
860   LPRINT USING "$$##.##";C;:
      LPRINT TAB(25) G
870   NEXT A
880  DATA 1.75,1.09,.682,.429,.269,.169,.107
890  LPRINT :
     LPRINT :
900  LPRINT "::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::::::::::::::::::"
910  LPRINT :
     LPRINT :
920  LPRINT "          PRICES SUBJECT TO CHANGE WITHOUT NOTICE"
930  LPRINT CHR$(12);
940  NEXT V
950 END
```

BUSINESS

Accountant's Aid

by James H. Sheats

Accountants, bookkeepers, analysts, and the rest of the number-scribbling fraternity spend much of their time with sheets of columnar work paper, printing a report title and date at the top, printing columnar headings across the page, and filling in lines and columns of figures. These sheets usually have both line and column totals, which should be equal when cross-footed. After considerable work with an adding machine and an eraser, they usually are.

The Accountant's Write-up Aid is designed to eliminate some of the pain associated with this process. You can adapt it to a number of purposes in its present state and customize it to your own needs. I wrote the program for a TRS-80 Level II, with a 132-column line printer (in my case, an IDS Paper Tiger). 4K of memory is plenty, since the program is very short—about 874 bytes.

The Program

The program uses both screen and printer output, but users without printers can eliminate all LPRINT statements and still have a useful program. The command on line 55, LPRINT CHR$ (31), is a Paper Tiger control command that adjusts the line length to 132 characters; it may not be necessary with other printers. Lines 65 and 68 allow you to input a report name and date. Both of these lines are unnecessary if you don't use a printer.

Line 70 is the reference name. This can be Date, Check Number, Invoice Number, or other suitable reference. This entry is not used for any computations.

In line 75, enter the number of columns that you want for a particular task. This version of the program is written with 10 characters per column and will print the reference column, data columns, and a line total column. (See Example 1.) You may customize your LPRINT USING statements for more or fewer columns. Lines 80 through 125 form a routine which inputs heading names for the columns.

In Lines 130 through 190, numerical data is entered. Each entry may be positive, negative, or zero. After accepting an entry for each column, the program prints a line total and repeats the routine for the next line. If you make an input error, you can reenter the same reference number, enter zero quantities for the the unchanged columns, and make corrections in the erroneous columns. The fourth day shown in Figure 1 illustrates this procedure. You can exit this program loop by entering a reference number of

999, or by establishing your own loop exit. Upon exit from the loop, the program prints column totals and a grand total. At that point, the program terminates. This routine is in lines 200 through 240.

NOVEMBER 1979

Date	Breakfast	Lunch	Dinner	Tips	Laundry	Postage	Phone	Supplies	Taxi	Plane	Room	Total
1	2.15	3.50	3.25	2.00	0.00	0.75	1.20	3.25	2.50	350.00	45.00	413.60
2	2.15	2.75	2.75	1.50	3.50	0.60	0.60	12.18	2.55	0.00	45.00	73.58
3	2.50	2.90	18.95	5.00	0.00	0.60	1.20	3.50	6.75	0.00	45.00	86.40
4	200.00	2.25	3.25	1.50	6.75	0.60	1.20	3.50	2.75	350.00	0.00	571.80
4	-200.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-200.00
4	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00
5	0.00	0.00	6.15	1.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.40

Example 1. *Travel expense summary*

One Disadvantage

One disadvantage to this program is that an entry must be made in each column, each time; however, typing a zero or ENTER will do. The program also does not provide heading routines for second and subsequent pages. Still, for the computer owner/businessperson, this simple program has a great deal of flexibiltiy and should be in his library along with amortization, checkbook balancing, depreciation, and other business programs.

Program Listing. *Accountant's Write-up Aid*

```
10 REM    "SPREAD SHEET"
20 REM    PROGRAMMED BY JAMES H. SHEATS
30 REM                  2036 HEADLAND DRIVE
40 REM                  EAST POINT, GA 30344
52 CLEAR 2000
53 DEFDBL C,T
55 CLS :
   LPRINT CHR$(31):
   LPRINT :
   LPRINT
60 PRINT "SPREAD SHEET PROGRAM"
65 INPUT "REPORT NAME";R$:
   LPRINT TAB(30)R$:
   LPRINT :
   LPRINT
68 INPUT "DATE";D$:
   LPRINT TAB(35)D$:
   LPRINT :
   LPRINT
70 INPUT "INPUT LINE NAME";L$
75 INPUT " NUMBER OF COLUMNS WANTED ";N
76 DIM C$(N),C(N),CT(N)
77 LPRINT USING "%              %"; L$;
80 FOR X = 1 TO N
90  INPUT "COLUMN NAME";C$(X)
100  LPRINT USING "%              %";C$(X);
120  NEXT
125 LPRINT "TOTAL"
130 PRINT L$:
    INPUT L
140 IF L = 999
     THEN
      200
144 LPRINT USING "###########";L;
145 T = 0
150 FOR X = 1 TO N
155  C(X) = 0
160  PRINT C$(X),:
     INPUT C(X):
     T = T + C(X)
165  LPRINT USING "#######.##";C(X);
170  CT(X) = CT(X) + C(X)
180  NEXT X
185 PRINT "TOTAL";T:
    LPRINT USING "######.##";T
190 GOTO 130
200 CLS :
    LPRINT :
    PRINT "TOTALS"; LPRINT "TOTALS     ";:
    FOR X = 1 TO N
204  PRINT C$(X),CT(X)
205  TT = TT + CT(X)
208  LPRINT USING "#######.##";CT(X);
210  NEXT X
220 PRINT "GRAND TOTAL ";TT
230 LPRINT USING "#######.##";TT
240 END
```

EDUCATION

Vocabulary Builder
Numerical Expression Input in Level II
Pre-School Math

Vocabulary Builder

by Roger Zimmerman

This program is designed primarily to help people build their knowledge of a foreign vocabulary, but it has other uses too. It can teach you the vocabulary of a specific discipline, as in biology or physics. It can even increase your English vocabulary.

Building Vocabulary

The first time you use this vocabulary program you will need to answer the keyboard-tape question with K, for keyboard. It will then ask you to enter a word and its definition. (Use *** to terminate a word.) You may enter either foreign words and English definitions or English words and foreign definitions or mix them. The word will be the unknown, and the definition will be the clue.

Allow the computer to quiz you until you are competent with your vocabulary list. It is helpful to pronounce the words as you type them to associate the pronunciation with the meaning. Look at the chart to see if there are any words that are stumbling blocks. You may wish to make a special tape of these difficult words. (The computer will ask you whether you want to do this.) If you want to make a tape of the entire list, answer N to the difficult words question and Y to the complete words question. Each time you return to the program, add two or three words to the list.

When you add new words after a session, the computer will ask if you wish to save the old list again. I have had no problem taping the new list right over the old list by starting the tape in about the same place. (My standard procedure is to start all computer tapes at counter 005.) When the list gets long, start a new one or weed out all of the easy words on the old lists for periodic review.

This program gives you 10000 bytes of available string space and a 100-item list limit. Note the use of CHR$(27) in lines 2015 and 2035. This ASCII code scrolls the cursor up one line. If you use CHR$(27) just before the INPUT data is printed on the screen, the new data will print where the prompt was. Then CHR$(30) must be used to remove the rest of the prompt. Here, CHR$(8) is used to remove the space between the entry number and the period.

Now look at line 1040. Note that the definition is printed on the screen first at TAB(23) plus one space. The cursor then returns to the beginning of the line and the word is printed. Sometimes the word is just long enough to push the left margin of the definition into the next field on the screen, but not long enough to cover it if it were to be left in the second field (assuming you use PRINT W$(I), D$(I)). Doing it as in line 1040 ensures that the left margin of the definition is always even, except when the word is really long. Then, the definition will begin just one space past the word.

You may also wonder about line 131. This allows the INKEY$ input to print in the right place. To have the computer turn on and off messages while waiting for an INKEY$ input, look at line 132. Usually the INKEY$ function works in a sort of scanning loop (i.e., 10 A$ = INKEY$:IFA$ = ""GOTO10). This way, if no input is encountered, it starts searching again. Suppose in between each scan we have the computer add 1 to a variable. Before it goes back for the next scan, it can increase the variable, and after the variable reaches a certain point, it can do whatever you'd like. (In this case, it turns off two of the lines on the screen.) Be careful not to cram too much in, or the computer might miss the input while doing the job.

How about getting the INKEY$ function to accept any length word? Look at line 133. Anything that is not a printable character, but is less than ASC$(32), will trigger a break from the search-for-another-letter routine. The ENTER key functions as it normally does with an INPUT statement. Otherwise the computer would look back and keep searching for another letter to add to the current string. Why didn't I just use an INPUT statement? Because I couldn't have turned off those lines while searching for an input.

One exception is that if the computer recognizes a CHR$(8), it goes to a special routine (line 138), the back-space arrow. This line counts the number of letters in the present input string and subtracts the last one. This doesn't work if there is just one character. The word would then become a "" (null string), and you would have to start the answer again.

On that list of difficult words, how do you keep from scrolling right off the screen when it is full? This sounds like a job for Super Machine Routine. Not so. If Radio Shack could do it in BASIC in their Budget Management program, I can do it in my Vocabulary Builder program. Radio Shack counts the number of lines it prints, then goes to the press-enter-to-continue routine.

That is not the way in which I approached this program. I remembered that the screen positions are treated as part of memory, and that PEEK could look and see what ASCII code was in any position of memory. If you PEEK at a spot where the bottom of the readout always changes the screen and test it to see if it is still blank, you'll know whether the screen is full. Look at line 7070.

Position 16305 is the spot on the screen where the first letter of the fourth column (field) of the last line you want is located. If PEEK(16305) ever becomes anything besides a blank space (CHR$(32)), then you'll know the screen is full and can then go to the enter-to-continue routine.

Program Listing. *Vocabulary Builder*

Encyclopedia Loader

```
  1 CLEAR 10000:
    DIM W$(100),D$(100),W(100),T(100)
 10 CLS :
    I = 1
 20 PRINT "FOREIGN LANGUAGE VOCABULARY PRACTICE"
 30 FOR X = 0 TO 1000:
     NEXT
 40 CLS
 50 PRINT "DO YOU WISH TO LOAD VOCABULARY BY TAPE OR BY KEYBOARD";
 60 A$ = INKEY$:
    IF A$ = "" GOTO 60
 70 IF A$ < > "T" AND A$ < > "K" GOTO 40
 80 IF A$ = "T" GOSUB 1000
 90 IF A$ = "K" GOSUB 2000
 99 II = I
100 CLS :
    PRINT "I WILL NOW GIVE YOU SOME DEFINITIONS AND ASK YOU TO GIVE
    ME THE CORRESPONDING VOCABULARY WORD. THE WORD MUST BE EXACT SPE
    LLING  AND MUST BE EXACT LEXICAL FORM."
105 PRINT :
    INPUT "PRESS ENTER TO CONTINUE";A
106 CLS
110 RANDOM :
    I = RND(II)
120 PRINT @0, CHR$(30):
    PRINT @0,D$(I)
125 PRINT CHR$(30);:
    PRINT "WHAT IS THE WORD?";:
    PRINT CHR$(30)
130 PRINT @896,"TO END TYPE 'E'"
131 PRINT @128,"":
    T$ = "":
    P = 0
132 S$ = INKEY$:
    P = P + 1:
    IF P = 250 PRINT @512, CHR$(30):
    PRINT @768, CHR$(30):
    GOTO 132:
     ELSE
      IF S$ = "" GOTO 132
133 IF ASC(S$) < 32 AND ASC(S$) < > 8 GOTO 136:
     ELSE
      U$ = "":
      IF ASC(S$) = 8 GOTO 138:
       ELSE
        GOTO 134
134 T$ = T$ + S$
135 PRINT @256, CHR$(30):
    PRINT @256,T$:
    GOTO 132
136 PRINT @256, CHR$(30):
    IF T$ = "E" GOTO 190
137 U$ = "":
    IF ASC(S$) = 8 GOTO 138:
     ELSE
      GOTO 140
138 PRINT CHR$(8):
    IF LEN(T$) = 1T$ = "":
    GOTO 135:
     ELSE
      PRINT @256, CHR$(30):
      TT = LEN(T$):
      FOR X = 1 TO TT - 1:
       U$ = U$ + MID$(T$,X,1):
       NEXT :
      T$ = U$:
      GOTO 135
140 IF T$ = W$(I) GOSUB 3000
```

Program continued

```
 150 IF T$ < > W$(I) GOSUB 4000
 160 GOTO 110
 190 GOSUB 7000
 200 CLS :
     IF A$ = "K" OR A$ = "Y" OR C = 1 PRINT "WOULD YOU LIKE TO SAVE T
     HE COMPLETE LIST OF WORDS YOU WERE DRILLED ON PLUS YOUR ADDITION
     S ON TAPE";
 210 B$ = INKEY$:
     IF B$ = "" GOTO 210
 220 IF B$ < > "Y" AND B$ < > "N" GOTO 200
 230 IF B$ = "Y" GOSUB 6000
 999 CLS :
     PRINT "BYE FOR NOW FROM THE VOCABULARY TESTER":
     FOR X = 0 TO 1000:
      NEXT :
     CLS :
     CLEAR 50:
     END
1000 CLS :
     INPUT "LOAD TAPE (PLAY) AND PRESS ENTER";A
1005 CLS :
     PRINT "LOADING VOCABULARY . . .PLEASE STAND BY"
1020 INPUT # - 1,W$(I),D$(I)
1028 IF W$(I) < > "***" GOTO 1040
1030 IF W$(I) = "***" PRINT "WOULD YOU LIKE TO ADD TO THIS LIST?"
1031 A$ = INKEY$:
     IF A$ = "" GOTO 1031
1032 IF A$ = "N"I = I - 1:
     RETURN
1033 IF A$ = "Y"C = 1:
     GOTO 50
1034 IF A$ < > "Y" AND A$ < > "N" GOTO 1031
1040 PRINT TAB(23)" "D$(I) CHR$(29)I CHR$(8)". "W$(I):
     I = I + 1
1050 GOTO 1020
2000 CLS
2010 INPUT "ENTER WORD ('***' TO END)";W$(I)
2015 PRINT CHR$(27) CHR$(30)I CHR$(8)". "W$(I)
2020 IF W$(I) = "***"I = I - 1:
     RETURN
2030 INPUT "ENTER DEFINITION";D$(I)
2033 IF PEEK(15384) = 32 PRINT
2035 PRINT CHR$(27) CHR$(27) CHR$(27)I CHR$(8)". "W$(I) TAB(23)" "D$(
     I) CHR$(30)
2040 I = I + 1
2050 PRINT :
     GOTO 2010
3000 R = R + 1
3002 PRINT @768, CHR$(30)
3005 T = T + 1
3007 T(I) = T(I) + 1
3010 RESTORE :
     Q = RND(10)
3020 FOR X = 1 TO Q:
      READ Q$:
      NEXT
3030 PRINT @512,Q$;:
     PRINT CHR$(30):
     PRINT @640,"THAT'S";R;"CORRECT OUT OF";T;"OR";(R * 100)
     / T;"%";:
     PRINT CHR$(30)
3040 IF T$ < > W$(I) PRINT @768,D$(I);" = ";W$(I);:
     PRINT CHR$(30)
3050 RETURN
4000 W = W + 1
4005 W(I) = W(I) + 1
4010 T = T + 1
4015 T(I) = T(I) + 1
4020 RESTORE :
     Q = 10 + RND(10)
4030 GOTO 3020
```

```
5000 DATA "RIGHT","CORRECT","VERY GOOD","YOU GOT IT","GOOD","ABSOLUTE
     LY RIGHT","THAT'S IT!","WHADDYA KNOW THAT'S RIGHT","NICE WORK -
     THAT'S IT","GOOD GOIN' CHARLIE, YOU'RE RIGHT","WRONG","INCORRECT
     ","OOPS","SORRY","STUDY THAT ONE SOME MORE","YOU GOOFED"
5010 DATA "NOPE","BITE YOUR TONGUE!","YOU SHOULD KNOW BETTER","DO YOU
      WANT TO KICK YOURSELF NOW?"

     PRINT "YOU WOULD LIKE TO SAVE ANY WORDS YOU ONLY GOT RIGHT --- %
      OF THETIME OR LESS. FILL IN THE BLANK.";:
     INPUT Q
5999 END
6000 CLS :
     INPUT "INSERT TAPE (RECORD) AND PRESS ENTER";Q
6001 CLS :
     PRINT "WRITING ONTO TAPE NOW"
6010 FOR I = 1 TO II:
      PRINT # - 1,W$(I),D$(I)
6012 PRINT I CHR$(8)". "W$(I), TAB(24)D$(I)
6015  NEXT
6020 PRINT # - 1,"***","***"
6030 RETURN
7000 CLS :
     PRINT "NOW I WILL SHOW YOU WHICH WORDS NEED THE MOST WORK."
7010 PRINT :
     PRINT "FOLLOWING IS A LIST OF EACH WORD YOU MISSED AT LEAST ONCE
            FOLLOWED BY THE NUMBER OF TIMES YOU MISSED IT AND OUT OF
     HOW    MANY TRIES."
7020 PRINT :
     INPUT "PRESS ENTER TO CONTINUE";A
7030 GOSUB 9000
7040 PRINT :
     FOR I = 1 TO II
7050  IF W(I) > 0 PRINT W$(I),W(I),T(I),((T(I) - W(I)) * 100)
      / T(I);"%"
7070  B = PEEK(16305):
      IF B < > 32 AND B + 1 < > 32 PRINT "PRESS ENTER TO CONTINUE";:
      INPUT A:
      GOSUB 9000
7080  NEXT
7082 INPUT "* END OF LIST *       PRESS ENTER TO CONTINUE";X
7092 PRINT :
     PRINT "NOW IF YOU WOULD LIKE TO SAVE ANY OR ALL OF THESE 'DIFFIC
     ULT' WORDS ON TAPE FOR LATER 'SPECIAL' REVIEW, YOU WILL THEN BE
     GIVEN A CHOICE OF THE LEVEL OF DIFFICULTY OF THOSE WORDS YOU WIS
     H TO SAVE."
7093 PRINT :
     PRINT "WOULD YOU LIKE TO SAVE ANY? (PRESS 'R' TO REVIEW LIST)"
7094 S$ = INKEY$:
     IF S$ = "" GOTO 7094
7095 IF S$ = "R" GOTO 7030
7096 IF S$ = "N" GOTO 200
7097 IF S$ < > "Y" AND S$ < > "N" AND S$ < > "R" GOTO 7094
7098 IF S$ = "Y" GOSUB 10000
8000 RETURN
9000 CLS :
     PRINT "WORD","TIMES MISSED","TIMES TRIED","% TIMES RIGHT":
     RETURN
10000 PRINT :
     PRINT "YOU WOULD LIKE TO SAVE ANY WORDS YOU ONLY GOT RIGHT --- %
      OF THETIME OR LESS. FILL IN THE BLANK.";:
     INPUT Q
10010 PRINT :
     INPUT "PREPARE TAPE (RECORD) AND PRESS ENTER";X
10015 CLS :
     PRINT "WRITING TAPE NOW"
10020 FOR I = 1 TO II:
      IF T(I) > 0 IF ((T(I) - W(I)) * 100) / T(I) < = Q PRINT W$(I),D
      $(I):
      PRINT # - 1,W$(I),D$(I)
10025  NEXT
```

Program continued

```
10027 PRINT # - 1,"***","***"
10030 RETURN
```

Numerical Expression Input in Level II

by Rodney Schreiner

A numerical expression is a combination of one or more constants, variables, and operations. (3 + A)*2 and EXP ((7.2 – X)/T) are examples of numerical expressions. Numerical expressions can be used in conjunction with several Level II BASIC statements, with all of the arithmetic functions, in FOR and IF statements, and with PRINT statements. When the Level II interpreter executes the statement PRINT 9 + 7, it evaluates the numerical expression 9 + 7 and displays the result, 16, on the screen. The Level II interpreter, however, does not accept a numerical expression as a response to an INPUT statement. For example, when the Level II interpreter executes the statement:

```
INPUT"HOW MANY INCHES ARE IN FIVE AND A HALF YARDS";A
```

a response of 5.5*3*12 will result in REDO? from the interpreter. Level II BASIC does not allow the use of a numerical expression as input.

The use of a numerical expression as input is helpful in educational science and mathematics programs. The purpose of such programs is often not to check the computational skills of students, but rather to test their comprehension of principles. When the testing of these basic principles involves several computations, an error in arithmetic is indistinguishable from a conceptual error to the computer. If the computer performs the calculation, any error is probably a conceptual one on the part of the student. Many students, when required to use a calculator while running an educational program, wonder why a computer can't tell that 2 + 2 is 4.

I have written a BASIC subroutine which permits the use of numerical expressions as input to a Level II program. The subroutine accomplishes this by treating the numerical expression as a string input and then inserting it into a line in the BASIC program where the interpreter translates it into the proper value. The subroutine is displayed in Program Listing 1.

To use the subroutine, the main program places the numerical expression input into the string variable A$. The subroutine is then called by GOSUB 65000. The subroutine returns with the value of the numerical expression in the variable AV. Program Listing 2 gives a short demonstration program which shows how the subroutine is used. Line 110 is the input statement which accepts the numerical expression and places it in A$. Line 120 calls the subroutine, which evaluates the numerical expression and returns its value in AV. Line 160 prints the value of the expression. Line 180 loops back for another input.

Lines 130 through 150 test error flags which the subroutine sets if there is a problem with the input. The subroutine, as listed, will handle expressions of up to 64 characters. If the input is longer than 64 characters, the subroutine returns with A$ equal to ERROR1. Line 130 checks for this condition. If A$ contains a character other than one which can be entered from the keyboard, the subroutine returns with A$ equal to ERROR2. Line 140 tests for this. If there is a syntax error in the input expression, the subroutine returns with A$ equal to ERROR3. This situation is assessed in line 150.

There are several restrictions in the use of the subroutine. The subroutine uses several variables in addition to A$ and AV; these are listed in line 65014. The subroutine changes the values of these variables; therefore, they should not be used in the main program. The variable QQ$ is of particular importance to the operation of the subroutine. It is essential that you not refer to QQ$ anywhere in a program other than in line 65028 of the subroutine. When you enter the subroutine into the computer, pay special attention to the format of lines 65028 and 65030. These lines are the heart of the subroutine and must be entered exactly as listed. In particular, there must be no spaces in line 65028; from the first Q to the final 8 there are exactly 18 characters followed by ENTER. Line 65030 must not have any spaces either. The AV = is followed by exactly 64 zeros. An easy way to determine when 64 zeros have been entered is to note that the last zero will be immediately under the the equal sign on the video display. (See Figure 1 for the formats of these lines.)

How it Works

The operation of the subroutine draws its power from a memory-saving feature of the Level II interpreter. The Level II interpreter stores a string that is defined in a BASIC line in the same instruction, rather than in high memory, so long as the value of the string is not changed. This allows you to use the VARPTR function of Level II to locate a particular BASIC line in memory. The subroutine does this then POKEs the input numerical expression into the program memory.

A line-by-line examination of the subroutine shows how it operates. Line 65020 defines the four numerical variables that the subroutine uses. If you use these variables anywhere else in the main program, the subroutine changes their values. This line also sets an error trap in case an error is generated in the subroutine. Line 65024 checks that the input string A$ is not too long. The length of the string is limited by the number of zeros you enter in line 65030. If an expression of more than 64 characters is to be used, you must insert more zeros in line 65030.

Line 65028 is the heart of the subroutine. This line contains the definition of the string variable QQ$. This variable is used only to locate this line in memory. Following the definition of QQ$ is a call to an internal subroutine.

LINE 65028

LINE POINTER		LINE NUMBER		Q	Q	$	=	"	*	"	:	GO-SUB	6	5	0	3	8	LINE END
XX	XX	4	254	81	81	36	61	34	42	34	58	145	54	53	48	51	56	0

LINE 65030

LINE POINTER		LINE NUMBER		A	V	=		
YY	YY	6	254	65	86	61		• • •

Figure 1. *Formats of lines 65028 and 65030*

This subroutine is located in lines 65038 through 65106 and serves to POKE the string A$ into program memory. It POKEs the string into program memory immediately following the AV = in line 65030. The 64 digits that follow the equal sign are merely a convenient way of reserving 64 bytes of program memory. At the time of execution, line 65028 calls an internal subroutine which changes the remainder of the line. Upon returning from the internal subroutine, the BASIC interpreter evaluates the numerical expression which is now in line 65030.

Line 65038, the first line of the internal subroutine, uses the VARPTR function to locate the byte which contains the asterisk in line 65028. The statement LP = VARPTR(QQ$) assigns the address of the byte which contains the length of the string QQ$ to the variable LP. Address LP + 1 contains the least significant byte, and address LP + 2 the most significant byte of the first character of the string QQ$; therefore, PEEK(LP + 1) + 256*PEEK(LP + 2) is the address of the asterisk in line 65028. The 16 bytes following the asterisk in line 65028 contain the codes for the characters up to the equal sign in line 65030. These bytes are shown in Figure 1. Upon completion of line 65038, LP holds the address of the byte containing the equal sign in line 65030. Line 65042 replaces the 64 digits in line 65030 with blank spaces (character code 32). Remember to change the 64 in line 65042 if you use other than 64 zeros in line 65030.

Lines 65046 through 65106 convert the string in A$ to character code and POKE the code into line 65030. Line 65046 converts each character in A$ to its character code. Line 65050 checks that the character code is that of a character entered from the keyboard. If it is not, A$ is set equal to ERROR2. Lines 65054 through 65062 convert character codes for arithmetic operation symbols to their respective operation codes. Lines 65066 through 65094 check for arithmetic functions. Line 65066 checks whether the remaining string is long enough to contain a function, and line 65068 checks whether the third character after the current character is a left parenthesis, as is required by a function. Lines 65072 through 65094 convert the function characters to the function code. Line 65102 skips the next two characters if a function is found in the string.

The error trapping routine, which checks for errors resulting from improper A$ input, is in line 65110. These input errors will result in a syntax error or a missing-operand error in line 65030. If such an error should occur in this line, the subroutine sets A$ equal to ERROR3. I chose the high line numbers of the subroutine so that they would not conflict with those of the main program. Once you have entered the subroutine from the keyboard, you can save it on tape. There are several ways to add the subroutine to the end of the main program. (See "APPEND It!" by Curtis F. Gerald in *80 Microcomputing*, February 1980, p.82.)

Program Listing 1. *Subroutine for numerical expression input*

Encyclopedia Loader

```
65000 :
      ' SUBROUTINE FOR NUMERICAL EXPRESSION INPUT
65002 :
      '   BY RODNEY SCHREINER
65004 :
      '
65006 :
      '    ENTER WITH NUMERICAL EXPRESSION IN A$
65008 :
      '    RETURN WITH VALUE OF NUMERICAL EXPRESSION IN AV
65010 :
      '      SUPPORTS ALL LEVEL II ARITHMETIC FUNCTIONS EXCEPT
65012 :
      '      TYPE CONVERSIONS: CDBL, CINT, AND CSNG
65014 :
      '          USES VARIABLES: LP, IX, CH, AV, A$, A9$, QQ$
65016 :
      '
65018 :
      ' CLEAR VARIABLES AND SET ERROR TRAP
65020 LP = 0:
      IX = 0:
      CH = 0:
      AV = 0:
      ON ERROR GOTO 65110
65022 :
      ' CHECK FOR MAXIMUM LENGTH OF A$ AS SET BY LINE 65030
65024 IF LEN(A$) > 64
       THEN
        A$ = "ERROR1" :
        RETURN
65026 :
      ' LOCATION FLAG AND CALL INTERNAL CONVERSION SUBROUTINE
65028 QQ$ = "*":
      GOSUB 65038
65030 AV=0000000000000000000000000000000000000000000000000000000
      000000000:' 64 ZEROES
65032 RETURN
65034 :
      ' INTERNAL CONVERSION SUBROUTINE ANALYZES A$
65036 :
      '     LOCATE AV
65038 LP = VARPTR(QQ$) :
      LP = PEEK(LP + 1) + 256 * PEEK(LP + 2) + 16
65040 :
      '     BLANK OUT OLD EXPRESSION IN AV
65042 FOR IX = 1 TO 64 :
       POKE LP + IX,32 :
       NEXT IX
65044 :
      '     LINES 65046 TO 65106 ANALYZE A$
65046 FOR IX = 1 TO LEN(A$) :
       A9$ = MID$(A$,IX,1) :
       CH = ASC(A9$)
65048  :
       '     CHECK FOR INPUT ERROR
65050  IF CH < 32 OR CH > 128
        THEN
         A$ = "ERROR2" :
         RETURN
65052  :
       '     CHANGE FROM CHARACTER CODE TO OPERATION CODE
```

Program continued

```
65054  IF A9$ = "*"
        THEN
         CH = 207 :
         GOTO 65098
65056  IF A9$ = "+"
        THEN
         CH = 205 :
         GOTO 65098
65058  IF A9$ = "-"
        THEN
         CH = 206 :
         GOTO 65098
65060  IF A9$ = "/"
        THEN
         CH = 208 :
         GOTO 65098
65062  IF A9$ = "["
        THEN
         CH = 209 :
         GOTO 65098
65064  :
       '    CHECK FOR ARITHMETIC FUNCTION
65066  IF IX + 5 > LEN(A$)
        THEN
         65098
65068  IF MID$(A$,IX + 3,1) < > "("
        THEN
         65098
65070  A9$ = MID$(A$,IX,3)
65072  IF A9$ = "EXP"
        THEN
         CH = 224 :
         GOTO 65098
65074  IF A9$ = "LOG"
        THEN
         CH = 223 :
         GOTO 65098
65076  IF A9$ = "SIN"
        THEN
         CH = 226 :
         GOTO 65098
65078  IF A9$ = "COS"
        THEN
         CH = 225 :
         GOTO 65098
65080  IF A9$ = "TAN"
        THEN
         CH = 227 :
         GOTO 65098
65082  IF A9$ = "SQR"
        THEN
         CH = 221 :
         GOTO 65098
65084  IF A9$ = "ATN"
        THEN
         CH = 228 :
         GOTO 65098
65086  IF A9$ = "SGN"
        THEN
         CH = 215 :
         GOTO 65098
65088  IF A9$ = "ABS"
        THEN
         CH = 217 :
         GOTO 65098
65090  IF A9$ = "INT"
        THEN
         CH = 216 :
         GOTO 65098
65092  IF A9$ = "RND"
```

```
        THEN
         CH = 222 :
         GOTO 65098
65094  IF A9$ = "FIX"
        THEN
         CH = 242 :
         GOTO 65098
65096  :
       '     POKE CODE INTO AV
65098  POKE LP + IX, CH
65100  :
       '     SKIP REST OF ARITHMETIC FUNCTION CHARACTERS
65102  IF CH > 210
        THEN
         IX = IX + 2
65104  NEXT IX
65106 RETURN
65108 :
      ' ERROR TRAPPING
65110 IF ERL = 65030
       THEN
        A$ = "ERROR3"
65112 RESUME NEXT
65114 END
```

Program Listing 2. *Demonstration of subroutine use*

```
100 CLEAR 500 :
    CLS
110 INPUT "WHAT IS THE NUMERICAL EXPRESSION"; A$
120 GOSUB 65000
130 IF A$ = "ERROR1" PRINT "INPUT LONGER THAN 64 CHARACTERS." :
    GOTO 110
140 IF A$ = "ERROR2" PRINT "ILLEGAL CHARACTER USED IN INPUT." :
    GOTO 110
150 IF A$ = "ERROR3" PRINT "SYNTAX ERROR IN NUMERICAL EXPRESSION." :
    GOTO 110
160 PRINT "THE VALUE OF ";A$;" IS ";AV
170 PRINT
180 GOTO 110
```

Pre-School Math

by Donald Hastings

Daddy, is that right?" My five-year old son showed real interest and aptitude in math, which I naturally wanted to encourage— but eventually that constant question can get to you—uh—me.

He was eager to know if his answer was right and just as eager to get some recognition for his efforts. I was tiring. I needed someone or something that did not tire yet still offered the recognition that he needed.

The following math program on the TRS-80 fulfilled that need admirably.

My son got the thrill of working Daddy's computer, the recognition he needed, and his math practice all at once. The program presses a youngster to grow by gradually giving him increasingly hard problems. These are mixed with easier ones. Also, the computer will back off if the operator is missing too many answers.

Operation

When a correct answer is given, the computer responds with a flashing "YES" and scores one point in the RIGHT column.

When an incorrect answer is given, the computer flashes a "NO," erases the response and gives the operator two more chances. If incorrect answers are entered all three times, the correct answer is displayed on the screen and one point is added to the WRONG column.

Every time the child gives five correct answers, the computer begins to increase the difficulty just a hair. If six incorrect answers are given, the difficulty is reduced slightly.

In this way, the computer adjusts to the level of the person working the problems while maintaining a slight upward pressure. After twenty problems the score is computed and flashed on the screen. If you haven't been watching and giving necessary encouragement to your youngster throughout the series, this is the point where you are called upon to bestow credit for achievement. You will also be able to see the level your math expert achieved and the final score.

If your child wishes to continue another series, the problems begin at a slightly higher level than where the previous series began, unless it proved to be too hard. By starting at a level below where the last series finished, the upward pressure is maintained, but the child is still encouraged.

A nice touch for the younger operators is that they have to learn to spell their name to play the game. Even though my five-year old son can't read the question, he knows his name must go there and promptly enters it to begin the series. He also knows what the RIGHT and WRONG columns are and can understand the flashing "YES" and "NO." Selecting the proper program and level from memory (ADDITION, LEVEL I—he hasn't gotten into subtraction yet) appears to give him no difficulty.

The degree of difficulty from one level to another is fairly subtle. If you find a particular level too simple, you may have to increase by several levels to find an appropriate one for your child.

Modifying the Program

It took a little experimenting to find out when to change levels and how many problems to include in a series without discouraging my son. Experiment with your child and modify the program to his needs. If you wish to increase the difficulty at a faster pace, line 7140 adjusts the level of difficulty down one when six incorrect answers are entered.

Note: this is six incorrect answers *entered*, not scored. To score six incorrect answers the person would have had to enter at least 18 incorrect ones. Also, a correct answer does not erase the record keeping of incorrect entries. If an operator consistently has to make several attempts at each problem, we might be pressing too hard.

Line 7150 moves the level up one for each five correct answers scored. This maintains the gradual degree of difficulty of the problems, but can be adjusted for your student.

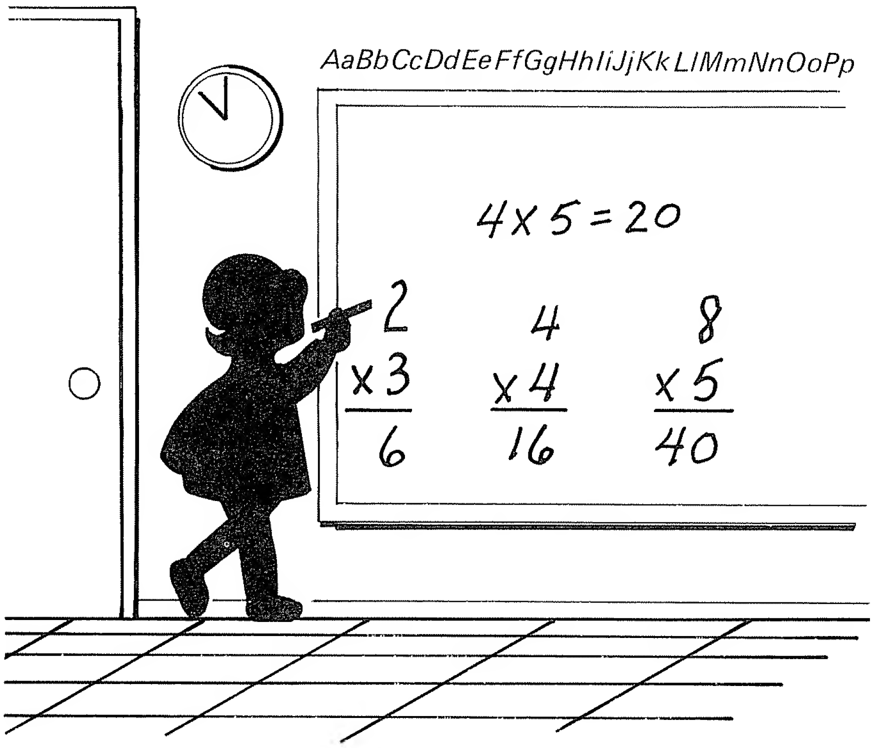
Line 7160 sets a series at 20 problems (T = 20) before the score is given. Adjust this figure to match the attention span of your child. You may even decide to make this a variable so that each series is different.

To adjust the number of times a problem can be missed before it is scored as incorrect, change the value of N in lines 7050, 4040, 3030, 2040, and 1030. The value must be the same in each line.

More Difficult Problems

Just how difficult your problems are going to be is determined by line 8030. The two numbers (A and B) are directly related to the level of difficulty (L) the operator has attained. To increase the average difficulty for each level, increase the value of the number added to L. (Be careful you don't go too quickly.)

The IF statement of line 8030 prevents the same problem from being presented twice in a row. If the problems involve division, line 8035 shunts the program to line 8050 where a new number is obtained for A that will not involve decimals when the division is performed. The answer to a division problem will always be an integer.



If the operator elects to continue in the same area of problems, line 9220 sets the starting level at three below the level just finished. Change the value subtracted from L if you want to vary this (the level cannot go below 1). When completing a series of 20 problems with a high score, the level of difficulty will advance through four levels. If you began in Level 1 and answered most of the problems correctly, on the next series you would begin in Level 2.

The program holds the interest of children of all ages. (The multiplication and division problems will be more to the liking of the advanced youngsters.) The combination of working a computer and seeing instant results prompts them to continue the series over and over. It would be an ideal addition to any primary school classroom.

Program Listing. *Pre-School Math*

Encyclopedia Loader

```
 10 REM   *** WRITTEN BY dON hASTINGS 4/79 ***
 20 REM   *** HEMINGWAY, s.c.  29554 ***
 30 REM   *** VARIABLES
 40 REM   *** A = 1ST #      B = 2ND #      C = CORRECT ANSWER
 50 REM   *** D = GUESS      L = LEVEL      M = CHOICE
 60 REM   *** N = NO'S       T = TURNS      S = SCORE
 70 REM   *** R = RIGHT      W = WRONG      U & V = CONTROL
100 CLS :
    FOR X = 1 TO 127:
     SET(X,1):
     NEXT
110 PRINT :
    PRINT :
    PRINT "   SELECTIONS:"
130 PRINT :
    PRINT :
    PRINT :
    PRINT ,"(A)  ADDITION":
    PRINT ,"(B)  SUBTRACTION":
    PRINT ,"(C)  MULTIPLICATION":
    PRINT ,"(D)  DIVISION"
140 FOR X = 1 TO 127:
     SET(X,32):
     SET(X,12):
     SET(X,44):
     NEXT
150 PRINT @ 770,"    CHOICE:"
160 FOR Y = 1 TO 44:
     SET(1,Y):
     SET(127,Y):
     NEXT
170 A$ = INKEY$:
    IF A$ = ""
     THEN
      170
180 M = ASC(A$) - 64
200 CLS :
    FOR X = 10 TO 100:
     SET(X,1):
     SET(X,12):
     NEXT
210 PRINT @ 138,"ENTER YOUR FIRST NAME:"
220 PRINT @ 404," ";:
    INPUT A$
230 PRINT @ 138,"ENTER STARTING LEVEL:  "
240 PRINT @ 404,"          ";:
    INPUT L
250 ON M GOTO 1000,2000,3000,4000
1000 CLS :
    PRINT TAB(20)"A  D  D  I  T  I  O  N":
    GOSUB 8000
1010 C = A + B:
    PRINT @ 537,A;"+";B;"= ";
1020 INPUT D:
    GOSUB 7000
1030 IF N < 3 GOTO 1010
1040 N = 0:
    GOSUB 8020:
    GOTO 1010
2000 CLS :
    PRINT TAB(16)"S  U  B  T  R  A  C  T  I  O  N":
    GOSUB 8000
2010 IF B > A
     THEN
      C = A:
      A = B:
      B = C
```

Program continued

```
2Ø2Ø C = A - B:
     PRINT @ 537,A;"-";B;"= ";
2Ø3Ø INPUT D:
     GOSUB 7ØØØ
2Ø4Ø IF N < 3 GOTO 2Ø2Ø
2Ø5Ø N = Ø:
     GOSUB 8Ø2Ø:
     GOTO 2Ø1Ø
3ØØØ CLS :
     PRINT TAB(1Ø)"M  U  L  T  I  P  L  I  C  A  T  I  O  N":
     GOSUB 8ØØØ
3Ø1Ø C = A * B:
     PRINT @ 537,A;"X";B;"= ";
3Ø2Ø INPUT D:
     GOSUB 7ØØØ
3Ø3Ø IF N < 3 GOTO 3Ø1Ø
3Ø4Ø N = Ø:
     GOSUB 8Ø2Ø:
     GOTO 3Ø1Ø
4ØØØ CLS :
     PRINT TAB(19)"D  I  V  I  S  I  O  N":
     GOSUB 8ØØØ
4Ø1Ø C = A / B
4Ø15 FOR X = 57 TO 66:
      SET(X,23):
      NEXT
4Ø2Ø PRINT @ 537,B;")";A;" = ";
4Ø3Ø INPUT D:
     GOSUB 7ØØØ
4Ø4Ø IF N < 3 GOTO 4Ø2Ø
4Ø5Ø N = Ø:
     GOSUB 8Ø2Ø:
     GOTO 4Ø1Ø
7ØØØ IF D = C
      THEN
       R = R + 1:
       V = V + 1:
       GOTO 71ØØ
7Ø1Ø FOR Z = 1 TO 5:
      PRINT @ 67Ø,"N O"
7Ø2Ø  FOR Y = 1 TO 1ØØ:
       NEXT Y
7Ø3Ø  PRINT @ 669,"     "
7Ø4Ø  FOR Y = 1 TO 1ØØ:
       NEXT Y:
      NEXT Z
7Ø5Ø N = N + 1:
     U = U + 1:
     IF N < 3 PRINT @ 547,"          ":
     RETURN
7Ø6Ø PRINT @ 67Ø,C:
     W = W + 1
7Ø7Ø FOR Z = 1 TO 2ØØØ:
      NEXT :
     PRINT @ 67Ø,"      ":
     RETURN
71ØØ FOR Z = 1 TO 3:
      PRINT @ 669,"Y E S"
711Ø  FOR Y = 1 TO 1ØØ:
       NEXT Y
712Ø  PRINT @ 669,"        "
713Ø  FOR Y = 1 TO 1ØØ:
       NEXT Y:
      NEXT Z
714Ø IF U > 5
      THEN
       L = L - 1:
       U = Ø:
       IF L < 1
      THEN
       L = 1
```

```
7150 IF V = 5
      THEN
       L = L + 1:
       V = 0
7160 N = 3:
     T = T + 1:
     IF T = 20 GOTO 9000
7170 RETURN
8000 FOR X = 17 TO 104:
      SET(X,3):
      NEXT :
     PRINT @ 153,"L E V E L "
8010 PRINT @ 265,"R I G H T":
     PRINT @ 300,"W R O N G"
8020 PRINT @ 164,L:
     PRINT @ 332,R:
     PRINT @ 367,W:
     PRINT @ 547,"          "
8030 A = RND(L + 4):
     B = RND(L + 3):
     IF A + B = E GOTO 8030
8035 E = A + B:
     IF M = 4 GOTO 8050
8040 RETURN
8050 A = B * RND(L + 2):
     RETURN
9000 S = R / (R + W):
     S = INT(S * 100):
     FOR X = 57 TO 66:
      RESET(X,23):
      NEXT
9010 PRINT @332,R:
     PRINT @367,W:
     PRINT @ 537,"                 "
9020 FOR X = 1 TO 127:
      SET(X,10):
      SET(X,18):
      SET(X,40):
      NEXT
9030 FOR Y = 10 TO 18:
      SET(63,Y):
      NEXT :
     FOR Z = 1 TO 200:
      NEXT
9100 PRINT @536,"  YOUR SCORE      "
9120 FOR X = 1 TO 10:
      PRINT @ 668,"     "
9130  FOR Y = 1 TO 100:
       NEXT Y
9140  PRINT @ 668,S
9150  FOR Y = 1 TO 100:
       NEXT Y:
      NEXT X
9160 PRINT @ 784,"WANT TO PLAY AGAIN, ";A$;:
     INPUT X$
9200 N = 0:
     T = 0:
     W = 0:
     U = 0:
     V = 0:
     R = 0
9210 IF ASC(X$) = 78 GOTO 100
9220 L = L - 3:
     IF L < 1
      THEN
       L = 1
9230 GOTO 250
```

GAMES

A Day at the Races

by James A. Swarts

One of the first programs I saw executed on the TRS-80 was a horse race program, similar to the one shown in Program Listing 1. As you can see by running this program, the horse race was little more than graphic dots moving across the video display.

Moving dots are okay for novice programmers, but they just aren't impressive to anyone else. This program needed a major face lift, and it got just that.

The first thing the program required was horses, and after several attempts, I finally managed to produce a decent-looking horse. There are really three horses (see Figure 1). This makes it appear as if the horses' legs move as they gallop down the field.

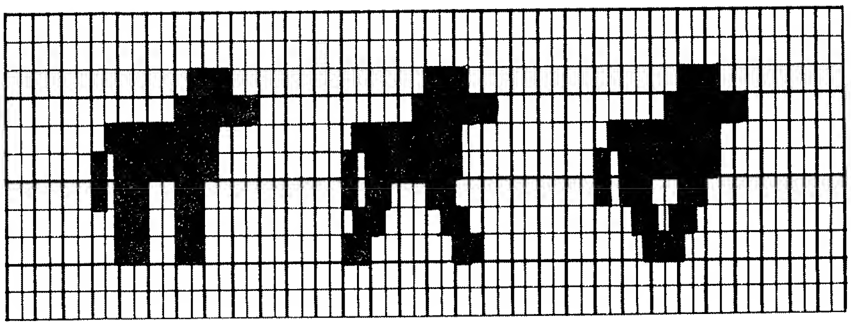


Figure 1

To be a real horse race, the program needed a provision for betting. At first, each player had a certain amount of money to bet and could only watch the races after the money was bet. You could only bet on the horse you thought would win the race; there were no payoffs for the second and third place horses. In my final version of the program, players can have negative earnings. I also allow for win, place, and show bets.

For those of you who haven't been to a race track, I'll explain the betting process. If you make a win bet, you get paid if the horse you bet on comes in first; if you make a place bet, you get paid if your horse finishes first or second; and if you make a show bet, you get paid if your horse finishes first, second, or third.

Payoffs are based on the odds of the horses, the number of bets made, the amount of the bets, and other such things. For simplicity, my Super Horse Race program uses randomly selected odds and other random factors to calculate payoffs.

SUPER HORSE RACE RACE 1

Number	Name	Odds
1	ASSEMBLY	51 : 1
2	FORTRAN	6 : 1
3	ALGOL	10 : 1
4	COBOL	4 : 1
5	PASCAL	3 : 1

Table 1. *Sample odds*

As an example, suppose the odds for the horses were those shown in Table 1. Assume that Pascal came in first, Assembly came in second, and Algol came in third. Payoffs would then be similar to those shown in Table 2. If you made a two-dollar bet on Assembly to place, you would get $100.20; if you made a two-dollar bet on Pascal to show, you would get $4.60; and so on. If you don't understand this, don't worry. The computer knows how much to pay you. All you need to do is type in the program in Program Listing 2, run it, and enjoy a day at the races.

SCHEDULE OF PAYOFFS
BASED ON A $2 BET

BET =>	Win	Place	Show
Finished	$ 6.00	$ 4.40	$ 4.60
2	______	$ 100.20	$ 102.00
3	______	______	$ 18.20

Table 2. *Payoffs based on sample odds*

Program Listing 1. *Original horse race program*

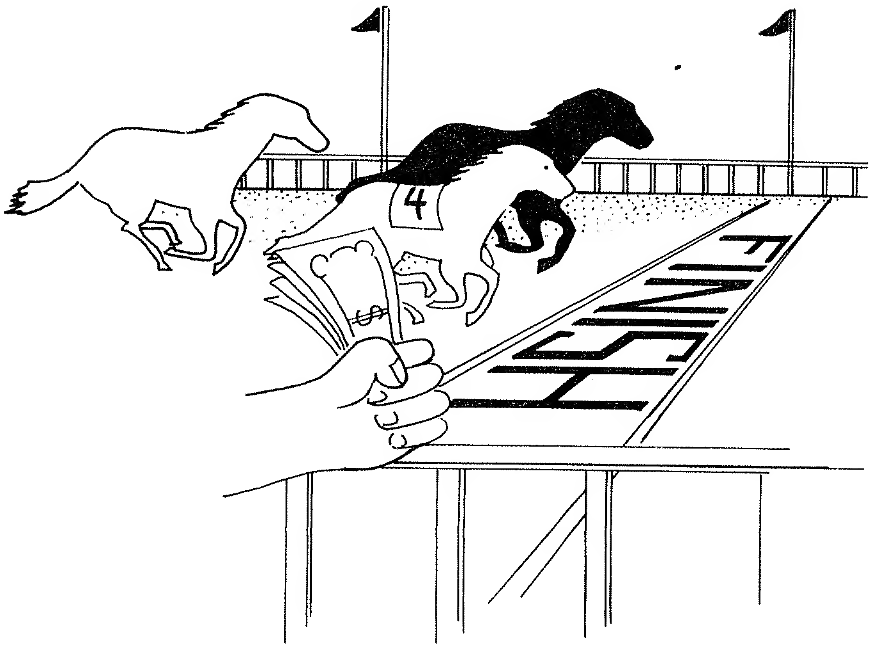
```
 10 CLEAR 100:
    RANDOM :
    C$ = STRING$(64,131)
 20 CLS :
    FOR L = 1 TO 7:
     PRINT C$:
     M(L) = 0:
     NEXT L
 30 FOR L = 0 TO 35:
     SET(124,L):
     NEXT L
 40 FOR L = 3 TO 33 STEP 6:
     SET(0,L):
     NEXT L
 50 I = RND(10):
    Q = RND(6):
    Y = Q * 6 - 3:
    RESET(M(Q),Y)
 60 M(Q) = M(Q) + I:
    IF M(Q) > 127
     THEN
      M(Q) = 127
 70 SET(M(Q),Y):
    IF M(Q) < 125
     THEN
      50
 80 PRINT "HORSE";Q;"WINS!":
    FOR L = 1 TO 1500:
     NEXT L
 90 W(Q) = W(Q) + 1:
    R = R + 1:
    CLS
100 FOR L = 1 TO 6:
     PRINT "HORSE ";L;" HAS WON";W(L);"RACES OR";W(L) * 100
     / R;"PERCENT":
     NEXT L
110 IF R = 10
     THEN
      130
120 FOR L = 250 TO 0 STEP - 1:
     PRINT @704,"THE NEXT RACE BEGINS IN"; INT(L / 25) + 1;"SECONDS
     ":
     NEXT L:
    GOTO 20
130 FOR L = 1 TO 2000:
     NEXT L:
    CLS
999 END
```

Program Listing 2. *Super Horse Race*

```
10 CLS :
   CLEAR 1000:
   RANDOM :
   GOTO 810
20 FOR T = 1 TO 25:
    NEXT T:
   RETURN
30 FOR T = 1 TO 500:
    NEXT T:
   RETURN
40 FOR T = 1 TO 1000:
    NEXT T:
```

Program continued

```
     RETURN
 50  I$ = INKEY$:
     IF I$ = ""
      THEN
       50:
       :
      ELSE
       RETURN
 60  CLS :
     FOR L = 0 TO 5:
      PRINT @L * 192, STRING$(63,131);:
      NEXT L
 70  FOR Y = 0 TO 45:
      SET(16,Y):
      SET(126,Y):
      NEXT Y
 80  PRINT @530,"THE HORSES ARE ENTERING THE STARTING GATE";
 90  FOR L = 0 TO 4:
      PRINT @L * 192,H$(1);:
      GOSUB 30:
      NEXT L
100  PRINT @530,". . . AND THEY'RE OFF !!!"; STRING$(16,32);:
     FOR T = 1 TO 300:
      NEXT T:
     PRINT @530, STRING$(25,32);
110  M = RND(5):
     IF F(M) = 1
      THEN
       110
120  P(M) = P(M) + 1:
     PA = 192 * (M - 1) + P(M)
130  PRINT @PA,H$(Q(P(M)));
140  IF P(M) < 55
      THEN
       110
150  FOR L = 1 TO 5:
```



```
     PRINT @PA,H$(1);:
     GOSUB 20
160  PRINT @PA,H$(4);:
     GOSUB 20:
     NEXT L
170 PRINT @PA + 68,C + 1;
180 F(M) = 1:
    C = C + 1:
    E(M) = C
190 IF C < 3
     THEN
      110:
      :
     ELSE
      C = 0
200 GOSUB 30:
    CLS :
    PRINT @534,"CALCULATING WINNINGS"
210 FOR L = 1 TO 5:
     A(L) = INT( RND(0) * 10) / 10:
     NEXT L
220 FOR L = 1 TO G:
     FOR LL = 1 TO 5:
      P(LL) = 0
230   IF (HB(L) = LL) AND (PB(L) = 1)
       THEN
        GOSUB 450
240   IF (HB(L) = LL) AND (PB(L) = 2)
       THEN
        GOSUB 470
250   IF (HB(L) = LL) AND (PB(L) = 3)
       THEN
        GOSUB 500
260   NEXT LL:
     NEXT L
270 CLS :
    PRINT "SCHEDULE OF PAY-OFFS":
    PRINT "BASED ON A $2 BET":
    PRINT
280 PRINT TAB(3);"BET =>"; TAB(13);"WIN"; TAB(28);"PLACE"; TAB(43);"
    SHOW":
    PRINT :
    PRINT "FINISHED":
    PRINT TAB(4);"1";
290 LL = 0:
    L = 0:
    E(L) = 1:
    W(L) = 2:
    FOR KK = 1 TO 5:
     IF E(KK) = 1
      THEN
       O(LL) = O(KK)
300  NEXT KK
310 DC(L) = 0:
    GOSUB 450:
    PRINT TAB(10); USING DS$;DC(L);
320 DC(L) = 0:
    GOSUB 470:
    PRINT TAB(25); USING DS$;DC(L);
330 DC(L) = 0:
    GOSUB 500:
    PRINT TAB(40); USING DS$;DC(L)
340 PRINT TAB(4);"2";:
    E(L) = 2:
    FOR KK = 1 TO 5:
     IF E(KK) = 2
      THEN
       O(LL) = O(KK)
350  NEXT KK:
    PRINT TAB(10); STRING$(10,45);
```

Program continued

```
360 DC(L) = Ø:
    GOSUB 48Ø:
    PRINT TAB(25); USING DS$;DC(L);
370 DC(L) = Ø:
    GOSUB 51Ø:
    PRINT TAB(4Ø); USING DS$;DC(L)
380 PRINT TAB(4);"3";:
    E(L) = 3:
    FOR KK = 1 TO 5:
     IF E(KK) = 3
      THEN
       O(LL) = O(KK)
39Ø  NEXT KK:
    PRINT TAB(1Ø); STRING$(1Ø,45);
4ØØ PRINT TAB(25); STRING$(1Ø,45);
41Ø DC(L) = Ø:
    GOSUB 52Ø:
    PRINT TAB(4Ø); USING DS$;DC(L)
42Ø PRINT :
    PRINT :
    PRINT "(PRESS ANY KEY TO CONTINUE)":
    GOSUB 5Ø
43Ø FOR L = 1 TO 5:
     F(L) = Ø:
     E(L) = Ø:
     NEXT L
44Ø RETURN
45Ø IF E(LL) = 1
     THEN
      DC(L) = DC(L) + O(LL) * W(L):
      :
     ELSE
      DC(L) = DC(L) - W(L)
46Ø RETURN
47Ø IF E(LL) = 1
     THEN
      DC(L) = DC(L) + (O(LL) - A(1)) * W(L):
      GOTO 49Ø
48Ø IF E(LL) = 2
     THEN
      DC(L) = DC(L) + (O(LL) - A(2)) * W(L):
      :
     ELSE
      DC(L) = DC(L) - W(L)
49Ø RETURN
5ØØ IF E(LL) = 1
     THEN
      DC(L) = DC(L) + (O(LL) - A(3)) * W(L):
      GOTO 53Ø
51Ø IF E(LL) = 2
     THEN
      DC(L) = DC(L) + (O(LL) - A(4)) * W(L):
      GOTO 53Ø
52Ø IF E(LL) = 3
     THEN
      DC(L) = DC(L) + (O(LL) - A(5)) * W(L):
      :
     ELSE
      DC(L) = DC(L) - W(L)
53Ø RETURN
54Ø RN = RN + 1:
    IF RN > 1Ø
     THEN
      77Ø
55Ø CLS :
    PRINT "S U P E R   H O R S E   R A C E"; TAB(5Ø);"RACE";RN:
    PRINT STRING$(64,131);
56Ø PRINT "NO."; TAB(15);"NAME"; TAB(35);"ODDS":
    PRINT :
    FOR L = 1 TO 5
```

```
570  IF RND(0) > .9
      THEN
       O(L) = RND(97) + 2:
       :
      ELSE
       O(L) = RND(8) + 2
580  R = RND(10):
     IF RF(R) = 1
      THEN
       580:
       :
      ELSE
       RF(R) = 1
590  S$ = STR$(O(L)):
     LN = LEN(S$):
     IF LN = 2
      THEN
       S$ = " " + S$
600  PRINT L; TAB(15);HN$(R); TAB(33);S$;" : 1"
610  NEXT L:
    PRINT :
    FOR L = 1 TO 10:
     RF(L) = 0:
     NEXT L
620 PRINT "ENTER WAGER, HORSE NUMBER, AND FINISHING POSITION"
630 FOR L = 1 TO G
640  PRINT @832,N$(L);:
     INPUT W(L),HB(L),PB(L)
650  IF ((W(L) < 2) OR (W(L) > 1000)) OR ((HB(L) < 1) OR (HB(L)
     > 5)) OR ((PB(L) < 1) OR (PB(L) > 3))
      THEN
       PRINT @832, STRING$(32,32):
       GOTO 640
660  PRINT @832, STRING$(32,32)
670  NEXT L:
    PRINT @832, CHR$(31):
    GOSUB 30
680 GOSUB 60
690 CLS :
    PRINT "WINNINGS AND LOSSES":
    PRINT :
    PRINT
700 FOR L = 1 TO G:
     PRINT N$(L); TAB(8);
710  IF DC(L) < 0
      THEN
       PRINT "HAS LOST $"; USING "#,########.##"; ABS(DC(L)):
       GOTO 730
720  PRINT "HAS WON  $"; USING "#,########.##";DC(L)
730  IF L / 10 = INT(L / 10)
      THEN
       PRINT @960,"(PRESS ANY KEY TO CONTINUE)";:
       :
      ELSE
       750
740  GOSUB 50:
     CLS
750  NEXT L:
    PRINT @960,"(PRESS ANY KEY TO CONTINUE)";:
    GOSUB 50
760 GOTO 540
770 CLS :
    PRINT "PRESS  1  TO PLAY AGAIN"
780 PRINT TAB(7);"2  TO STOP GAME":
    GOSUB 50
790 IF I$ = "1"
     THEN
      RUN
800 IF I$ = "2"
     THEN
```

Program continued

```
        CLS :
        GOTO 9999:
        :
       ELSE
        GOTO 77Ø
 81Ø DS$ = "$#,####.##":
     S$ = STRING$(9,32):
     PRINT @538,"H O R S E"
 82Ø FOR L = 1 TO 7:
      D = L * 64 + 78:
      PRINT @D,"S U P E R";
 83Ø  PRINT @1Ø76 - D,"R A C E":
      FOR T = 1 TO 1ØØ:
       NEXT T
 84Ø  PRINT @D,S$;:
      PRINT @1Ø76 - D,S$:
      NEXT L
 85Ø PRINT @D,"S U P E R";:
     PRINT @1Ø76 - D,"R A C E":
     GOSUB 4Ø
 86Ø PRINT @588,"B Y   J A M E S   A .   S W A R T S"
 87Ø DIM Q(55):
     FOR L = 1 TO 55:
      READ Q(L):
      NEXT L
 88Ø DATA 1,2,1,3,1,2,1,3,1,2,1,3,1,2,1,3,1,2,1,3,1,2
 89Ø DATA 1,3,1,2,1,3,1,2,1,3,1,2,1,3,1,2,1,3,1,2,1,3
 9ØØ DATA 1,2,1,3,1,2,1,3,1,2,1
 91Ø FOR L = 1 TO 1Ø:
      READ HN$(L):
      NEXT L
 92Ø DATA "FORTRAN","COBOL","BASIC","RPG","ASSEMBLY"
 93Ø DATA "ALGOL","PASCAL","PL/I","SNOBOL","JOVIAL"
 94Ø FOR L = 1 TO 4
 95Ø  READ D:
      IF D = - 1
       THEN
        97Ø
 96Ø  H$(L) = H$(L) + CHR$(D):
      GOTO 95Ø
 97Ø  NEXT L
 98Ø DATA 131,131,131,131,163,179,26,24,24,24,24,24,24,32,152,188,188
     ,191,191,131,26,24,24,24,24,24,24,24,32,129,191,32,17Ø,149,-1
 99Ø DATA 131,131,131,131,163,179,26,24,24,24,24,24,24,32,152,188,188
     ,191,191,131,26,24,24,24,24,24,24,24,32,185,135,32,13Ø,173,144,-
     1
1ØØØ DATA 131,131,131,131,163,179,26,24,24,24,24,24,24,32,152,188,188
     ,191,191,131,26,24,24,24,24,24,24,24,32,129,139,18Ø,158,129,-1
1Ø1Ø DATA 131,131,131,131,131,131,26,24,24,24,24,24,24,32,32,32,32,32
     ,32,32,26,24,24,24,24,24,24,24,32,32,32,32,32,32,32,-1
1Ø2Ø CLS :
     PRINT @526,"DO YOU REQUIRE INSTRUCTIONS (Y OR N)?"
1Ø3Ø GOSUB 5Ø:
     IF I$ = "N"
      THEN
       112Ø:
       :
      ELSE
       IF I$ < > "Y"
        THEN
         1Ø3Ø
1Ø4Ø CLS :
     PRINT "S U P E R   H O R S E   R A C E"
1Ø5Ø PRINT "I N S T R U C T I O N S":
     PRINT
1Ø6Ø PRINT "THERE WILL BE 1Ø RACES WITH FIVE HORSES RUNNING IN  EACH R
     ACE."
1Ø7Ø PRINT "WHEN YOU ARE PROMPTED TO PLACE YOUR BET, ENTER THE  FOLLOW
     ING:"
```

```
1080 PRINT "   (1) THE AMOUNT YOU WISH TO BET ($2 - $1000)":
     PRINT "   (2) THE HORSE YOU WISH TO BET ON (1 - 5)":
     PRINT "   (3) WIN, PLACE, OR SHOW (1 - 3)":
     PRINT
1090 PRINT "IF THE HORSE YOU BET ON WINS THE RACE, YOU WILL BE PAID I
     F":
     PRINT "YOU MADE A WIN, PLACE, OR SHOW BET.  IF YOUR HORSE COMES
     IN":
     PRINT "SECOND, YOU WILL BE PAID IF YOU MADE A PLACE OR SHOW BET.
     "
1100 PRINT "IF YOUR HORSE FINISHES THIRD, YOU WILL BE PAID IF YOU MAD
     E":
     PRINT "A SHOW BET."
1110 PRINT :
     PRINT "(PRESS ANY KEY TO CONTINE)";:
     GOSUB 50
1120 CLS :
     INPUT "HOW MANY PLAYERS (1 - 30)";G
1130 IF (G < 1) OR (G > 30)
      THEN
       1120:
       :
      ELSE
       DIM N$(G),DC(G),HB(G),W(G),PB(G)
1140 PRINT :
     FOR L = 1 TO G:
      PRINT "ENTER NAME OF PLAYER";L;:
      INPUT N$(L)
1150  LN = LEN(N$(L)):
      IF LN > 6
       THEN
        N$(L) = LEFT$(N$(L),6)
1160  NEXT L:
     GOTO 540
9999 END
```

Star Dreck

by Delton T. Horn

I think it's an unwritten law that every programmer must, at some time, write a Star Trek program. My version, Star Dreck, is both an exciting game and a spoof. The player is the captain of the starship Boobyprize. Throughout the game, crew members kibitz the captain for his inept performance against the rival Klingons.

The object of the game is to destroy a specific number of Klingon ships. The computer determines the number depending on the game level you enter in line 30. The computer will accept any number from 1 to 50, but since levels 1 through 10 would be too easy for you, the computer will reset these levels to level 50.

On each turn you have nine command options. Engaging the warp engines moves the ship through the galaxy. You must enter the speed and distance of travel you want. If your request is too large and would overload the warp engines, the command is refused. The ship can move fore (away from base) or aft (towards base), but must never leave the confines of the galaxy or it will be destroyed. If the ship falls into a black hole, the computer relocates it somewhere in the galaxy.

You can fire either phasers or photon torpedoes to destroy Klingons. If you apply too little or too much power to the phasers or try to fire an empty photon torpedo bank, your command is refused. Phaser power must be within the range of 15 to 410 units. Each time you return to base, you may replace one torpedo, but you can never have more than 10.

Command number 4 returns the ship to base for repairs. The warp engines require several star dates to recover, but other repairs are completed within a single star date. The Klingons also repair their ships when you return to base. If you use ordinary warp drive to dock at base, no repairs are made. As long as a ship is docked at base it has an inexhaustible energy supply. The number of star dates available for play is determined by the number of Klingons in your sector.

You can also choose to take a status report (see Figure 1) or a scan of the nearest Klingon (distance and energy level). The Klingons slowly approach the base, but the firing angle never changes. Additional commands include disciplining crew members, the self-destruct command, and ordering a security report in case of a traitor on board. Just as in real life, your security team occasionally makes a mistake.

The Star Dreck program (see Program Listing) was written for a 32K Level II Model III, and is slightly too large for a 16K machine. If you

```
                                                              14:52:33
         STATUS REPORT—STARSHIP BOOBYPRIZE—STAR DATE ** 12

                    CAPTAIN DELTON COMMANDING

            CURRENT LOCATION          257
            CREW                    395         SPOCK IS DEAD
            ENERGY LEVEL            9974
            LIFE SUPPORT            94.2932%
            SHIELDS                 82%
            WARP ENGINES            98%
            PHOTON TORPEDOES                    10
            KLINGON SHIPS IN PATROL AREA        10
```

Figure 1. *Sample status report*

eliminate the line numbers shown in Table 1 and change line 25 to CK = 0:CX = 0, it will fit. This deletes the traitor on board routine and the security report. The program should also run on a Level II Model I.

Even without the humorous comments from the crew, the game is fun to play. There are a number of surprises hidden within the program, and you'll learn the secrets of successful play as you play more games. But don't worry that you'll get bored when you learn all the details. I wrote it and I still lose to the Klingons occasionally.

Eliminate the following lines for a 16K machine.
27
217
240
305
527
1097
2600-2620
3330-3385
3420-3497
3570-3795
4000-4060
4200
4210

Table 1. *Modifications for a 16K machine*

Program Listing. *Star Dreck*

Encyclopedia Loader™

```
  5 REM * STAR DRECK *
  7 REM * BY DELTON T. HORN *
  8 REM * COPR. JUNE, 1981 *
 10 DIM KD(100):
    DIM KA(100)
 15 CLS :
    PRINT :
    PRINT
 20 NV = 100:
    CP = 100:
    FOR X = 1 TO 20:
     KD(X) = 0:
     NEXT X
 25 CK = 0:
    CX = 0:
    TX = RND(7):
    IF TX > 3
     THEN
      TX = 0:
     ELSE
      TX = 1
 27 IF TX = 1 GOSUB 3330
 30 INPUT "GAME LEVEL";GL
 35 IF GL > 50 GOTO 30
 40 IF GL < 10
     THEN
      GL = 50
 50 T = GL * 10 + 30 + RND(50)
 60 SP = 1:
    SC = 1:
    MC = 1:
    CH = 1:
    SU = 1:
    LU = 1
 65 KL = RND(GL) + 5:
    Y = RND(500) + 100
 70 FOR X = 1 TO KL:
     KD(X) = Y:
     Y = Y + RND(50) + 25
 75  KA(X) = RND(330) + 15:
     NEXT X
 80 SD = 1:
    LC = 0:
    WE = 10
100 CLS :
    PRINT :
    PRINT :
    PRINT " ","STAR DRECK":
    PRINT :
    PRINT :
    PRINT
110 INPUT "CAPTAIN'S NAME";N$
130 IF (N$ = "KIRK") OR (N$ = "JAMES KIRK") OR (N$ = "JAMES T. KIRK"
    ) OR (N$ = "JIM KIRK") GOTO 1500
160 CLS :
    PRINT :
    PRINT " ","STAR DRECK":
    PRINT :
    PRINT "SPOCK: WELCOME ABOARD THE BOOBYPRIZE, CAPTAIN ";N$
165 PRINT :
    PRINT :
    PRINT
170 EN = 10000:
    PT = 10:
    CR = 400:
    LS = 100:
    SH = 100:
    BT = 0:
```

```
    IC = 0
180 INPUT "INSTRUCTIONS";I$
185 IF LEFT$(I$,1) = "Y" GOSUB 2500
190 KK = KL
195 PRINT "THERE ARE ";KL;" KLINGON SHIPS IN YOUR SECTOR!"
200 INPUT E$:
    CLS :
    PRINT :
    PRINT " ","STAR DATE ** ";SD:
    GOSUB 1800
201 IF CR < 2 GOTO 1535
202 IF TR$ = "SCOTTY"
     THEN
      TR$ = "SCOTT"
205 IF WE < 9.5
     THEN
      WE = WE + RND(5) / 100
207 FOR XX = 1 TO 222:
     NEXT XX
208 GG = RND(30):
    IF (GG > 20) AND (TX = 1) GOSUB 4200
210 IF SD > T GOTO 1200:
     ELSE
      PRINT "    COMMAND CHOICES":
      IF LC = 0
       THEN
        EN = 10000
212 NK = NK - RND(8):
    IF NK = LC
     THEN
      NK = NK - 1
213 IF NK < 2 GOSUB 3300
214 IF NK < 1 GOTO 217
215 IF LC = 0 PRINT "The BOOBYPRIZE is docked at BASE"
217 IF TX = 1 GOSUB 3500
220 PRINT "1 --- Engage warp engines":
    PRINT "2 --- Fire phasers"
225 PRINT "3 --- Fire photon torpedo":
    PRINT "4 --- Return to BASE"
230 PRINT "5 --- Status report":
    PRINT "6 --- Scan Klingons"
235 PRINT "7 --- Enforce on board discipline":
    PRINT "8 --- Self-destruct"
240 PRINT "9 --- Request security report"
280 PRINT
290 PRINT "WHAT IS YOUR COMMAND, CAPTAIN ";N$;
300 INPUT C:
    SD = SD + 1
305 IF C = 9 GOTO 3570
310 IF C = 1 GOTO 450
320 IF C = 2 GOTO 550
330 IF C = 3 GOTO 880
340 IF C = 4 GOTO 910
350 IF C = 5 GOTO 925
360 IF C = 6 GOTO 1020
370 IF C = 7 GOTO 1050
380 IF C = 8 GOTO 1700
390 IC = IC + 1:
    PRINT "ILLOGICAL COMMAND!":
    IF IC > 10 GOTO 400:
     ELSE
      GOTO 200
400 PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT "MESSAGE FROM FEDERATION":
    PRINT
```

Program continued

```
405 PRINT " ","DISCIPLINARY BOARD TO CAPTAIN ";N$:
    PRINT :
    PRINT
410 PRINT "Your repeated illogical orders prove you to be unfit for"
415 PRINT "command!":
    PRINT :
    GOSUB 1000:
    PRINT "You are demoted to ensign and confined to quarters!!"
420 PRINT :
    PRINT :
    PRINT :
    END
450 INPUT "WARP SPEED";S:
    S = INT(S):
    IF S > 10 GOTO 1570
460 IF S < 1 GOTO 390
470 INPUT "DISTANCE";D:
    D = INT(D):
    IF D > 30 * WE GOTO 1570
475 IF D < 1 GOTO 390
477 IF NV < 1 GOTO 3050
480 INPUT "FORE OR AFT";DR$:
    D$ = LEFT$(DR$,1):
    IF D$ = "F" GOTO 525
490 IF D$ = "A" GOTO 520
495 NV = NV - 0.5
500 PRINT "NOW, WHICH WAY DO YOU WANT TO GO, CAPTAIN ";N$;"?":
    GOTO 480
520 D = 0 - D
525 LC = LC + D:
    IF LC < 0 GOTO 1550
527 IF LC = KD(NQ) GOTO 2600
530 NK = ABS(LC - KD(NQ)):
    IF LC > 11000 GOTO 1550
535 EN = INT(EN - D * S / 10):
    IF EN < 1 GOTO 1530
540 GOTO 200
550 INPUT "POWER TO PHASER BANKS";P
560 IF P > 410 GOTO 1570
565 IF P < 15 GOTO 1680
567 EN = EN - P
570 INPUT "FIRING ANGLE";A
575 A = INT(A):
    IF A < 0 GOTO 390
580 IF A > 360 GOTO 1690
585 IF (A < 10) OR (A > 350) GOTO 700
590 IF (A < KA(NQ) + 7) AND (A > KA(NQ) - 7) GOTO 815
595 IF NK > 400 PRINT "WHY DID YOU FIRE INTO EMPTY SPACE?":
    GOTO 610
597 IF ABS(A - KA(NQ)) > 75 PRINT "   WAY OFF TARGET!",
600 PRINT "   MISS!"
610 Z = RND(14)
615 IF (Z < 4) AND (SP = 0) GOTO 610
620 IF (Z > 3) AND (Z < 7) AND (SU = 0) GOTO 610
630 IF (Z = 7) AND (MC = 0) GOTO 610
635 IF (Z > 9) AND (CH = 0) GOTO 610
640 IF Z = 1 PRINT "SPOCK:That really stinks, Captain."
645 IF Z = 2 PRINT "SPOCK:Is that the best you can do, Captain?"
650 IF Z = 3 PRINT "SPOCK:I'll bet that really scared the socks off
    'em."
655 IF Z = 4 PRINT "SULU:You blew it, sir."
660 IF Z = 5 PRINT "SULU:May I have a transfer, Captain?"
665 IF Z = 6 PRINT "SULU:Have your eyes always been crossed, Captain
    ?"
670 IF Z = 7 PRINT "McCOY:I think you need a long rest, Captain!"
675 IF Z = 10 PRINT "CHEKOV:Mama mia!"
680 IF Z = 11 PRINT "CHEKOV:Why don't you try a squirt gun next time
    , Captain?"
685 IF Z = 12 PRINT "CHEKOV:Maybe I should've joined the Klingons."
690 IF Z = 13 PRINT "CHEKOV:I want some vodka!"
```

```
695 IF Z = 14 PRINT "CHEKOV:Pretty bad, Captain!"
697 GOTO 200
700 PRINT "YOU JUST FIRED ON YOUR OWN SHIP!":
    R = 0:
    DC = RND(9):
    DX = RND(9) + 1:
    CR = CR - DX
705 PRINT DX;" CREW MEMBERS HAVE BEEN KILLED!"
710 IF (DC = 1) AND (SP = 1) GOTO 750
715 IF (DC = 2) AND (SC = 1) GOTO 755
720 IF (DC = 3) AND (MC = 1) GOTO 760
725 IF (DC = 4) AND (SU = 1) GOTO 765
730 IF (DC = 5) AND (CH = 1) GOTO 770
735 IF (DC = 6) AND (LU = 1) GOTO 775
740 IF R = 1 GOTO 2050
742 IF R = 2
     THEN
      RETURN
745 GOTO 200
750 SP = 0:
    PRINT "SPOCK ";:
    GOTO 780
755 SC = 0:
    PRINT "SCOTTY ";:
    GOTO 780
760 MC = 0:
    PRINT "McCOY ";:
    GOTO 780
765 SU = 0:
    PRINT "SULU ";:
    GOTO 780
770 CH = 0:
    PRINT "CHEKOV ";:
    GOTO 780
775 LU = 0:
    PRINT "LT. UHURA ";
780 PRINT "IS DEAD.":
    IF R = 1 GOTO 2080
782 IF R = 2
     THEN
      RETURN
785 GOTO 200
815 PRINT "       DIRECT HIT!!!"
820 G = P - NK / 2
825 IF G < 5
     THEN
      G = 5
830 KS = KS - RND(G):
    GOSUB 1000
835 IF KS < 0 GOTO 860
840 GOTO 200
860 KS = 0:
    KD(NQ) = 0:
    K = K - 1
865 NK = 0:
    KD(NQ) = 0:
    IF CH = 0 GOTO 875
870 PRINT "CHEKOV REPORTS THE KLINGON SHIP IS DESTROYED ";:
    VV = RND(3):
    IF VV = 2 PRINT "THEN PASSES OUT":
     ELSE
      PRINT
872 GOTO 200
875 PRINT "KLINGON SHIP IS DESTROYED!":
    GOTO 200
880 IF PT > 0 GOTO 900
885 PRINT "YOU HAVE USED UP ALL OF YOUR PHOTON TORPEDOES, CAPTAIN ";
    N$:
    IF SP = 0 GOTO 200
887 CM = RND(4):
```

Program continued

```
     PRINT "SPOCK:You are really ";:
     IF CM = 1 PRINT "stupid";
 890 IF CM = 2 PRINT "a turkey";
 894 IF CM = 3 PRINT "useless";
 896 IF CM = 4 PRINT "illogical";
 898 PRINT ", CAPTAIN ";N$;".":
     GOTO 200
 900 PT = PT - 1:
     P = 500:
     GOTO 570
 910 IF WE < 5 GOTO 1300
 911 IF NK < 1 GOTO 5000
 912 LC = 0:
     EN = 10000:
     IF PT < 10
      THEN
       PT = PT + 1
 915 LS = 100:
     SH = 100:
     BT = BT + 1:
     NK = 0:
     N = 0
 920 NV = NV + 10:
     IF BT > 10 GOTO 1010
 925 CLS :
     PRINT :
     PRINT "   STATUS REPORT --- STARSHIP BOOBYPRIZE --- STAR DATE **
      ";SD
 930 PRINT " ","CAPTAIN ";N$;" COMMANDING"
 935 IF LC = 0 PRINT "SHIP IS DOCKED AT BASE",
 940 PRINT "CURRENT LOCATION   ";LC:
     PRINT "CREW",CR,:
     IF SP = 0 PRINT "SPOCK IS DEAD",
 942 IF SC = 0 PRINT "SCOTTY IS DEAD",
 944 IF CH = 0 PRINT "CHEKOV IS DEAD",
 946 IF SU = 0 PRINT "SULU IS DEAD",
 948 IF MC = 0 PRINT "McCOY IS DEAD",
 950 IF LU = 0 PRINT "UHURA IS DEAD",
 955 PRINT :
     PRINT "ENERGY LEVEL",EN
 960 PRINT "LIFE SUPPORT",LS;"%":
     PRINT "SHIELDS",SH;"%"
 965 PRINT "WARP ENGINES",WE * 10;"%":
     PRINT "PHOTON TORPEDOES",PT
 967 IF R = 4 GOTO 2700
 968 IF R = 10 GOTO 2200
 970 K = 0:
     FOR X = 1 TO KL:
      IF KD(X) > 0
       THEN
        K = K + 1
 980  NEXT X:
     PRINT "KLINGON SHIPS IN PATROL AREA",K
 990 GOTO 200
1000 FOR X = 1 TO 333:
      NEXT X
1005 RETURN
1010 PRINT "YOU HAVE MADE TOO MANY TRIPS BACK TO BASE, ";N$:
     GOTO 410
1020 IF (NK = 0) OR (NK > 500) GOTO 1035
1025 PRINT "NEAREST KLINGON VESSEL IS ";NK;" PARSECS AWAY":
     PRINT "OPERATING AT ";KS;"% ENERGY LEVEL":
     GOTO 200
1035 PRINT "NO KLINGONS WITHIN RANGE":
     GOTO 200
1050 INPUT "CREW MEMBER TO BE DISCIPLINED";C$:
     CR = CR - 1
1060 IF (C$ = "SPOCK") OR (C$ = "MR SPOCK")
      THEN
       SP = 0
1070 IF (C$ = "MCCOY") OR (C$ = "DR MCCOY") OR (C$ = "BONES")
```

```
      THEN
       MC = 0
1080 IF (C$ = "SCOTTY") OR (C$ = "MR SCOTT")
      THEN
       SC = 0
1085 IF (C$ = "CHEKOV") OR (C$ = "MR CHEKOV")
      THEN
       CH = 0
1090 IF (C$ = "SULU") OR (C$ = "MR SULU")
      THEN
       SU = 0
1095 IF (C$ = "UHURA") OR (C$ = "LT UHURA")
      THEN
       LU = 0
1097 GOSUB 3420
1100 IF C$ = N$ GOTO 1150
1105 D$ = "CAPTAIN":
     E$ = " ":
     F$ = D$ + E$ + N$
1107 IF (C$ = D$) OR (C$ = F$) GOTO 1150
1110 CK = CK + 1:
     PRINT C$;" IS DEAD.":
     IF CK > 10 GOTO 3810:
      ELSE
       GOTO 200
1150 PRINT "WHY DID YOU COMMIT SUICIDE?":
     FOR X = 1 TO 2000:
      NEXT X:
     NEW
1200 PRINT "YOU HAVE FOOLED AROUND FOR ";SD;" STAR DATES, CAPTAIN ";N
     $:
     GOSUB 1000
1210 PRINT "AFTER ALL THAT TIME, THE KLINGONS HAVE DEFEATED THE FEDER
     ATION"
1220 GOSUB 1000:
     GOTO 1540
1300 PRINT "YOUR WARP ENGINES ARE BADLY DAMAGED!":
     GOTO 200
1500 PRINT "DON'T GET CUTE, PAL!":
     GOSUB 1000:
     PRINT "YOU ARE MOST CERTAINLY NOT CAPTAIN KIRK!":
     GOTO 110
1530 PRINT "THE BOOBYPRIZE HAS NO ENERGY IN RESERVE!"
1535 PRINT "THE ENTIRE CREW IS DECEASED!":
     GOSUB 1000
1540 PRINT "THANKS A LUMP, 'CAPTAIN' ";N$
1545 R = 10:
     GOTO 940
1550 PRINT "THE BOOBYPRIZE JUST WANDERED OUT OF THE KNOWN GALAXY":
     GOSUB 1000:
     PRINT "ALL OF YOUR STAR MAPS ARE COMPLETELY USELESS HERE":
     GOSUB 1000
1560 GOTO 1535
1570 IC = IC + 0.25:
     Z = RND(15):
     Y = 0:
     IF (Z < 4) AND (SP = 0) GOTO 1570
1580 IF (Z > 6) AND (Z < 10) AND (SC = 0) GOTO 1570
1590 IF (Z > 9) AND (Z < 13) AND (SU = 0) GOTO 1570
1600 IF (Z > 12) AND (CH = 0) GOTO 1570
1605 IF Z = 1 PRINT "SPOCK:That is not a logical choice, Captain."
1610 IF Z = 2 PRINT "SPOCK:You are not quite sane, Captain."
1615 IF Z = 3 PRINT "SPOCK:Whoo boy!  Humans are such dips!"
1620 IF (Z > 3) AND (Z < 7) GOTO 390
1625 IF Z = 7 PRINT "SCOTTY:The engines couldn't stand the strain, si
     r!"
1630 IF Z = 8 PRINT "SCOTTY:Have you been getting into my scotch agai
     n, Captain?"
1635 IF Z = 9 PRINT "SCOTTY:Oy vey!"
1640 IF Z = 10 PRINT "SULU:Are you sure, Captain?"
```

Program continued

```
1645 IF Z = 11 PRINT "SULU:Have you flipped, Captain?"
1650 IF Z = 12 PRINT "SULU:May I have a transfer, Captain?"
1655 IF Z = 13 PRINT "CHEKOV:May I have a transfer, Captain?"
1660 IF Z = 14 PRINT "CHEKOV:Tee hee hee!"
1665 IF Z = 15 PRINT "CHEKOV:You have a hole in your head, Captain!"
1670 PRINT :
     PRINT :
     GOSUB 1000:
     GOTO 200
1680 IF SP = 0 GOTO 550
1685 PRINT "SPOCK:That ain't diddly-poo, Captain.":
     GOTO 1670
1690 A = INT(A / 360):
     GOTO 580
1700 T = RND(100) + 50:
     CLS :
     FOR X = 1 TO T:
      Y = RND(1010):
      PRINT @Y," * ";:
      NEXT X
1730 PRINT "   KA-BOOM!!!!!":
     PRINT :
     PRINT :
     GOSUB 1000:
     GOTO 1540
1800 J = 0:
     H = 1000000:
     FOR X = 1 TO KL:
      J = KD(X) + J:
      IF SD > T GOTO 1200
1805  G = ABS(KD(X) - LC):
      IF G = LC GOTO 1815
1810  IF G < H
       THEN
        H = G:
        NQ = X
1815  NEXT X:
     IF J = 0 GOTO 2150
1817 N = NQ
1820 IF (NK = 0) OR (NK > H) GOTO 1840
1830 QX = RND(222) + 222:
     IF (NK > 0) AND (NK < QX) GOTO 1850
1832 IF (NK > 0) AND (NK < 425) PRINT "KLINGON VESSEL SIGHTED!"
1835 GOSUB 1000:
     PRINT :
     RETURN
1840 NK = H:
     KS = 100:
     GOTO 1830
1850 PRINT "KLINGON VESSEL SIGHTED!":
     GOSUB 1000:
     PRINT "THEY FIRE UPON THE BOOBYPRIZE!"
1860 KF = RND(400) - NK:
     GOSUB 1000:
     IF KF > 100 GOTO 2000
1875 PRINT "  THEY MISS!"
1880 CM = RND(9):
     IF (CH = 0) AND (CM < 4) GOTO 1880
1885 IF (SU = 0) AND (CM > 5) GOTO 1880
1890 IF CM = 1 PRINT "CHEKOV:Hallelujah!"
1892 IF CM = 2 PRINT "CHEKOV:I need some vodka!"
1894 IF CM = 3 PRINT "CHEKOV:Now what are you going to do, Captain
1896 IF CM = 6 PRINT "SULU HIDES UNDER YOUR CHAIR AND QUIVERS"
1898 IF CM = 7 PRINT "SULU:I WANT MY MOMMY!!!!"
1900 IF CM = 8 PRINT "SULU:May I have a transfer, Captain?"
1902 IF CM = 9 PRINT "SULU:I don't think they like us, Captain"
1910 RETURN
2000 FOR X = 15360 TO 16380:
      POKE X,42:
      NEXT X
2005 PRINT "THE BOOBYPRIZE HAS BEEN HIT!!"
```

```
2010 FR = ABS(NK - LC) / 3:
     IF FR > 90
      THEN
       FR = 80
2015 DM = RND(100 - FR):
     X = RND(DM):
     DM = DM - X:
     Y = RND(DM):
     Z = DM - Y:
     Z = ABS( INT(Z)):
     IF Z < 2
      THEN
       Z = 2
2020 PRINT "   YOU SUFFER DAMAGE!":
     SH = SH - X:
     IF SH < - 10 GOTO 2130
2025 LS = LS - Y:
     IF LS < 1 GOTO 2120
2030 CR = CR - Z:
     IF CR < 2 GOTO 1535
2035 IF MC = 1 PRINT "McCOY:We lost ";Z;" crew members!"
2040 IF (SC = 1) AND (SH < 25) PRINT "SCOTTY:The shields are in very
     bad shape, sir!"
2045 IF (LU = 1) AND (LS < 25) PRINT "LT UHURA:The life support syste
     ms have been badly damaged!"
2047 R = 1:
     DC = RND(15):
     GOTO 710
2050 WE = WE - 0.1:
     CM = RND(27):
     IF CM = 8
      THEN
       WE = WE - RND(8) / 2
2051 IF (SU = 0) AND (CM = 27) GOTO 2050
2052 IF (CM < 4) AND (SP = 0) GOTO 2050
2053 IF (CM > 7) AND (CM < 12) AND (SC = 0) GOTO 2050
2054 IF (CM = 12) AND (LU = 0) GOTO 2050
2056 IF (CM > 15) AND (CM < 22) AND (CH = 0) GOTO 2050
2057 IF CM = 16
      THEN
       CX = CX + ( RND(300) / 100)
2058 IF CX > 10 GOTO 3800
2060 IF (CM > 23) AND (CM < 27) AND (MC = 0) GOTO 2050
2062 IF (CM = 3) OR (CM = 5)
      THEN
       CP = CP - ( RND(3000) / 1000)
2063 IF CP < 1 GOTO 3000
2064 IF CM = 1 PRINT "SPOCK:My tummy is upset."
2066 IF CM = 2 PRINT "SPOCK:"I BELIEVE WE ARE IN TROUBLE, CAPTAIN."
2068 IF CM = 3 PRINT "SPOCK:The computer has sustained damage!"
2070 IF CM = 8 PRINT "SCOTTY:The warp engines have been damaged!"
2072 IF CM = 9 PRINT "SCOTTY:It's looking bad, Captain!"
2074 IF CM = 10 PRINT "SCOTTY:My bottle of scotch is broken!"
2076 IF CM = 11 PRINT "SCOTTY:The engines are under a severe strain,
     sir!"
2077 IF (CM = 11) OR (CM = 4)
      THEN
       EN = EN - INT(EN / 4)
2078 IF CM = 12 PRINT "LT UHURA:The communications system has been da
     maged, sir!"
2080 IF CM = 13 PRINT "SULU:May I have a transfer, Captain?"
2082 IF CM = 14 PRINT "SULU:The navigation system has been damaged!"
2083 IF (CM = 14) OR (CM = 7)
      THEN
       NV = NV - RND(5)
2084 IF CM = 15 PRINT "SULU:Just wait until I get my hands on that re
     cruiting officer!"
2086 IF CM = 27 PRINT "SULU:I stubbed my toe!"
2088 IF CM = 16 PRINT "CHEKOV:The coffee maker was damaged!"
2090 IF CM = 17 PRINT "CHEKOV:I hit my funny bone!"
2092 IF CM = 19 PRINT "CHEKOV:May I have a transfer, Captain?"
```

Program continued

```
2094 IF CM = 20 PRINT "CHEKOV:I should've gone to dental school!"
2096 IF CM = 21 PRINT "CHEKOV:My bottle of vodka is broken!"
2100 IF CM = 24 PRINT "McCOY:";N$;"!  You have to stop this!"
2102 IF CM = 25 PRINT "McCOY:Nurse Chappel skinned her knee!"
2104 IF CM = 26 PRINT "McCOY:I fell on a hypo and an overdose!"
2106 IF CM = 26
      THEN
       MC = 0
2110 R = 2:
     DC = RND(27):
     IF DC < 7 GOTO 710
2115 RETURN
2120 PRINT "THE LIFE SUPPORT SYSTEM HAS BEEN COMPLETELY DESTROYED!":
     GOTO 1535
2130 PRINT "THE SHIELDS HAVE BEEN COMPLETELY DESTROYED!":
     PRINT :
     GOSUB 1000
2135 PRINT "THE BOOBYPRIZE IS QUICK-FRIED TO A CRACKLY CRUNCH!":
     GOSUB 1000:
     GOTO 1535
2150 PRINT "YOU HAVE DESTROYED ALL OF THE KLINGONS IN YOUR PATROL SEC
     TOR!"
2160 GOSUB 1000:
     PRINT "GOOD JOB, ";:
     GOSUB 1000
2170 PRINT "ADMIRAL ";N$;"!!"
2180 R = 10:
     GOTO 940
2200 PRINT "STAR DATE *** ";SD
2210 PRINT "YOUR MISSION WAS TO DESTROY ";KK;" KLINGONS"
2220 END
2500 PRINT "Your mission is to destroy all of the Klingon ships in yo
     ur"
2505 PRINT "patrol sector.  You may use either phasers or photon torp
     edoes."
2510 PRINT "You have only a limited number of torpedoes.  Using the"
2515 PRINT "phasers depletes your energy reserve. All damage to Kling
     ons"
2520 PRINT "(and to you!) is cumulative.  You may refresh your energy
     , life"
2525 PRINT "support and shields by returning to base.  This will also
      give"
2530 PRINT "the Klingons a chance to recover.  Too many trips back to
      base"
2535 PRINT "or invalid commands will result in your immediate demotio
     n."
2540 FOR X = 1 TO 333:
      NEXT X
2545 PRINT "Incidentally, once you have located a Klingon ship, it wi
     ll not"
2550 PRINT "move.  Press 'ENTER' after each command.  A question mark
      and"
2555 PRINT "flashing square will prompt you for a command.  Your crew
      will"
2560 PRINT "offer helpful comments as you go along.  Press 'ENTER' to
      play";
2565 INPUT G$
2570 RETURN
2600 CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT
2610 PRINT "  YOU JUST RAMMED INTO A KLINGON SHIP!"
2620 GOSUB 1000:
     GOSUB 1000:
     GOTO 1700
2700 PRINT " ","P.";:
     GOSUB 1000:
     PRINT "U."
```

```
2710 END
2999 STOP
3000 GOSUB 1000:
     CLS :
     PRINT :
     PRINT :
     PRINT "THE COMPUTER IS DESTROYED!":
     FOR X = 1 TO 2000:
      NEXT X
3010 NEW
3050 PRINT "THE BADLY DAMAGED NAVIGATION SYSTEM FORCES YOU TO MOVE IN
      A":
     PRINT "RANDOM DIRECTION!":
     DR = RND(2)
3055 IF DR = 1 GOTO 525
3060 GOTO 520
3300 PRINT "THE KLINGONS HAVE INVADED YOUR STARBASE!":
     T = T - RND(10):
     IF LC = 0 GOTO 3400
3310 RETURN
3330 TR = RND(11):
     IF TR = 1TR$ = "CLANCY"
3335 IF TR = 2TR$ = "SULU"
3340 IF TR = 3TR$ = "SPOCK"
3345 IF TR = 4TR$ = "McCOY"
3350 IF TR = 5TR$ = "CHEKOV"
3355 IF TR = 6TR$ = "WILSON"
3360 IF TR = 7TR$ = "BROWN"
3365 IF TR = 8TR$ = "UHURA"
3370 IF TR = 9TR$ = "SQUIBBLES"
3375 IF TR = 10TR$ = "KLAGSTORN"
3380 IF TR = 11TR$ = "SCOTTY"
3385 CL = 1:
     WL = 1:
     BR = 1:
     SQ = 1:
     KG = 1:
     RETURN
3400 PRINT "THE BOOBYPRIZE IS OVERRUN WITH KLINGONS!":
     GOSUB 1000
3410 PRINT "YOU ARE DEFEATED!":
     R = 10:
     GOTO 940
3420 IF C$ = "SCOTT"
      THEN
       SC = 0
3422 IF C$ = "CLANCY"
      THEN
       CL = 0
3425 IF C$ = "WILSON"
      THEN
       WL = 0
3430 IF C$ = "BROWN"
      THEN
       BR = 0
3435 IF C$ = "SQUIBBLES"
      THEN
       SQ = 0
3440 IF C$ = "KLAGSTORN"
      THEN
       KG = 0
3445 IF (TR = 1) AND (CL = 0)
      THEN
       TX = 0
3450 IF (TR = 2) AND (SU = 0)
      THEN
       TX = 0
3455 IF (TR = 3) AND (SP = 0)
      THEN
       TX = 0
3460 IF (TR = 4) AND (MC = 0)
```

Program continued

```
       THEN
        TX = 0
3465 IF (TR = 5) AND (CH = 0)
       THEN
        TX = 0
3470 IF (TR = 6) AND (WL = 0)
       THEN
        TX = 0
3475 IF (TR = 7) AND (BR = 0)
       THEN
        TX = 0
3480 IF (TR = 8) AND (LU = 0)
       THEN
        TX = 0
3485 IF (TR = 9) AND (SQ = 0)
       THEN
        TX = 0
3490 IF (TR = 10) AND (KG = 0)
       THEN
        TX = 0
3495 IF (TR = 11) AND (SC = 0)
       THEN
        TX = 0
3497 RETURN
3500 TQ = RND(5):
      IF TQ > 3 GOTO 3520
3510 RETURN
3520 TQ = RND(4):
      IF TQ = 1
       THEN
        EN = EN - RND(500)
3530 IF TQ = 2
       THEN
        LS = LS - RND(10)
3540 IF TQ = 3
       THEN
        SH = SH - RND(10)
3550 IF TQ = 4
       THEN
        WE = WE - RND(4) / 2
3560 RETURN
3570 PRINT :
      PRINT " ","SECURITY REPORT":
      PRINT "STARSHIP BOOBYPRIZE"," ","STAR DATE ** ";SD:
      GOSUB 1000
3575 PRINT :
      GOSUB 1000:
      TQ = RND(100):
      IF (TX = 1) AND (TQ < 98) GOTO 3590
3580 IF (TX = 0) AND (TQ > 98) GOTO 3590
3585 PRINT "ALL CLEAR, CAPTAIN":
      GOTO 200
3590 PRINT "SABATOGE SUSPECTED --- A TRAITOR MAY BE ON BOARD!":
      PRINT
3595 PRINT " ","COMMAND CHOICES":
      PRINT "1 --- Prepare suspect list (1 extra star date)":
      PRINT "2 --- Individual security report (3 extra star dates)":
      PRINT "3 --- Cancel security alert"
3600 INPUT "YOUR COMMAND";C:
      IF C = 1 GOTO 3620
3610 IF C = 2 GOTO 3700
3615 IF C = 3 GOTO 200:
       ELSE
        GOTO 390
3620 PRINT :
      PRINT :
      GOSUB 1000:
      PRINT "CREW MEMBERS SUSPECTED OF POSSIBLE TREASON":
      PRINT :
      GOSUB 1000:
      SD = SD + 1
```

```
3625 QM = 1:
     GOSUB 4000:
     IF (QT = 11) AND (BR = 1) PRINT "BROWN",
3630 QM = 2:
     GOSUB 4000:
     IF (QT = 11) AND (CH = 1) PRINT "CHEKOV",
3635 QM = 3:
     GOSUB 4000:
     IF (QT = 11) AND (CL = 1) PRINT "CLANCY",
3640 QM = 4:
     GOSUB 4000:
     IF (QT = 11) AND (KG = 1) PRINT "KLAGSTORN",
3645 QM = 5:
     GOSUB 4000:
     IF (QT = 11) AND (MC = 1) PRINT "MCCOY",
3650 QM = 6:
     GOSUB 4000:
     IF (QT = 11) AND (SC = 1) PRINT "SCOTT",
3655 QM = 7:
     GOSUB 4000:
     IF (QT = 11) AND (SP = 1) PRINT "SPOCK",
3660 QM = 8:
     GOSUB 4000:
     IF (QT = 11) AND (SQ = 1) PRINT "SQUIGGLES",
3665 QM = 9:
     GOSUB 4000:
     IF (QT = 11) AND (SU = 1) PRINT "SULU",
3670 QM = 10:
     GOSUB 4000:
     IF (QT = 11) AND (LU = 1) PRINT "UHURA",
3675 QM = 11:
     GOSUB 4000:
     IF (QT = 11) AND (WL = 1) PRINT "WILSON",
3680 PRINT :
     PRINT :
     GOSUB 1000:
     GOTO 3595
3700 PRINT "CREW MEMBER TO BE INVESTIGATED?  (ENTER LAST NAME ONLY --
      NO":
     INPUT "NICK-NAMES OR TITLES) ";SP$
3710 GOSUB 1000:
     GOSUB 1000:
     SD = SD + 3:
     IF SP$ = TR$ GOTO 3795
3720 PRINT SP$;" DOES NOT APPEAR TO BE THE CULPRIT":
     GOSUB 1000
3725 QT = RND(10):
     IF QT > 5 GOTO 200
3730 PRINT "OFFENDED BY YOUR DISTRUST, ";SP$;" COMMITS SUICIDE"
3735 IF SP$ = "BROWN"
      THEN
       BR = 0
3740 IF SP$ = "CHEKOV"
      THEN
       CH = 0
3745 IF SP$ = "CLANCY"
      THEN
       CL = 0
3750 IF SP$ = "KLAGSTORN"
      THEN
       KG = 0
3755 IF SP$ = "MCCOY"
      THEN
       MC = 0
3760 IF SP$ = "SCOTT"
      THEN
       SC = 0
3765 IF SP$ = "SPOCK"
      THEN
       SP = 0
3770 IF SP$ = "SQUIBBLES"
```

Program continued

```
      THEN
       SQ = 0
3775 IF SP$ = "SULU"
      THEN
       SU = 0
3780 IF SP$ = "UHURA"
      THEN
       LU = 0
3785 IF SP$ = "WILSON"
      THEN
       WL = 0
3790 CR = CR - 1:
     GOTO 200
3795 PRINT :
     PRINT TR$;" IS A TRAITOR!!!!":
     GOSUB 1000:
     PRINT :
     GOTO 200
3800 PRINT "THE COFFEE POT IS DESTROYED!!!":
     GOSUB 1000
3810 PRINT "THE CREW MUTINIES!":
     GOSUB 1000
3820 PRINT "YOU WILL BE PUT TO DEATH AT MIDNIGHT!":
     GOSUB 1000
3830 R = 10:
     GOTO 940
4000 QT = RND(10):
     IF QT > 5
      THEN
       QT = 11
4005 IF (QM = 1) AND (TR$ = "BROWN")
      THEN
       QT = 11
4010 IF (QM = 2) AND (TR$ = "CHEKOV")
      THEN
       QT = 11
4015 IF (QM = 3) AND (TR$ = "CLANCY")
      THEN
       QT = 11
4020 IF (QM = 4) AND (TR$ = "KLAGSTORN")
      THEN
       QT = 11
4025 IF (QM = 5) AND (TR$ = "McCOY")
      THEN
       QT = 11
4030 IF (QM = 6) AND (TR$ = "SCOTT")
      THEN
       QT = 11
4035 IF (QM = 7) AND (TR$ = "SPOCK")
      THEN
       QT = 11
4040 IF (QM = 8) AND (TR$ = "SQUIGGLES")
      THEN
       QT = 11
4045 IF (QM = 9) AND (TR$ = "SULU")
      THEN
       QT = 11
4050 IF (QM = 10) AND (TR$ = "UHURA")
      THEN
       QT = 11
4055 IF (QM = 11) AND (TR$ = "WILSON")
      THEN
       QT = 11
4060 RETURN
4200 PRINT "ONE OF YOUR CREW MEMBERS IS FOUND MURDERED!"
4210 CR = CR - 1:
     RETURN
5000 PRINT "THE KLINGONS CAPTURE YOU ALONG WITH THE BASE!"
5010 R = 4 GOTO 940
```

GRAPHICS

Slide Show
Graphs, Plus

Slide Show

by Tom Van Dan Elzen

The Etch-A-Screen concept has simplified and created new possibilities for computer graphics. But, for people who would like to use graphics more creatively, Slide Show offers more. Slide Show has three unique twists that give it expanded audiovisual capability:
1) Ten full-screen slides are CSAVEd as part of the program.
2) An assembly-language routine that is POKEd into place when the program is RUN prints any one of the 10 slides, on command, in milliseconds.
3) A program that allows a time delay between slides. (If you select zero time between slides, you can have up to a 10-frame "movie.")
Credit where credit is due. The actual drawing routine and text routine are taken from "Etch-A-Screen" (James K. Shrum, *80 Microcomputing*, May 1980).

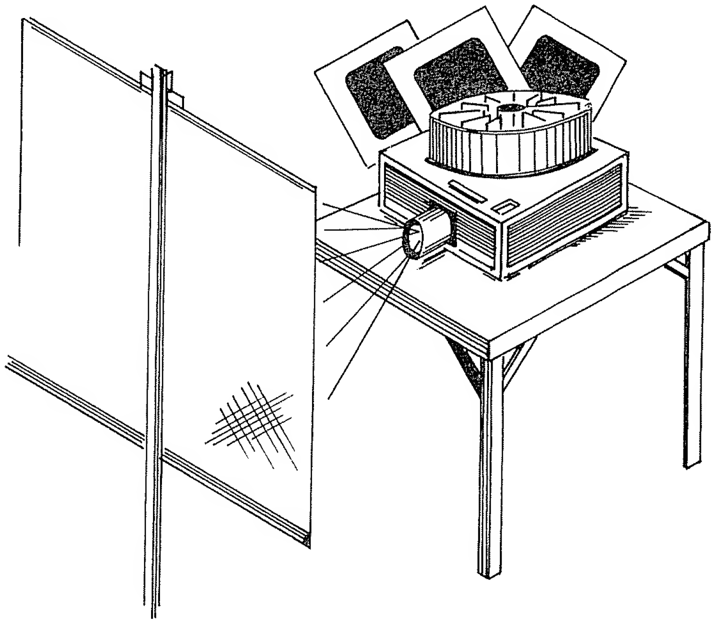
The Program

The slides are stored in lines 1 through 50 which consist of "dummy" REMark statements containing 204 characters per line. Each slide is stored consecutively, with lines 1 through 5 storing slide one, 6 through 10 storing slide two, etc. For now, ignore line 51; we'll get to it later. But remember that lines 1 through 50 must have exactly 204 characters per line.

Routine 1000 initializes the system and POKEs the assembly routine (the DATA statement) into memory starting at 7C00 hex. The six commands that the program uses are then printed on the CRT to refresh your memory. Line 1260 is an INKEY routine (PEEK 16537) that is looking for the CLEAR key to be pressed (31). If the key is pressed, the A from line 51 is POKEd into the lower right of the CRT, and the system is sent to subroutine 4000 (TEXT).

The workings of subroutines 2000, 3000, and 4000 are adapted from Etch-A-Screen, as mentioned above. The heart of the program, the graphics, is accomplished in routine 2000. Routine 3000 directs the program to the different subroutines you select. Routine 4000 writes the text of the program. Slides are cleared from memory and replaced with asterisks in routine 5000. Because we store the entire slide, you can store a new slide over an old slide, if you wish, without clearing the old slide. If you do this, you can delete all of the 5XXX lines and line 3140.

Routine 6000 stores the slide, byte by byte, into the proper lines (1 through 50) as designated by the slide number. If you store a slide in memory and then list the program, don't be shocked by what you see in the first 50 lines. These strange appearances are byte codes for graphics (128–196) that are transformed into words by the BASIC interpreter like KILL, GOTO, IF, THEN, etc. If you CSAVE the program, it will all save.



Slides are redrawn on the CRT by subroutine 7000. The X = USR(V) command passes the start location of the desired slide (variable V) up to the assembly-language routine. For slide zero, the start location is V = 17136; for slide one it is V = 17136 + 1060 (1060 = 5 × 204 + 40). The slide is stored in five lines with 204 bytes per line. (This is only 1020 bytes because we don't save the last four bytes in the lower right corner of the CRT.) The added 40 bytes are the overhead created by the line numbers and the REMark(') in each line. I have not seen any other program that can CSAVE data to tape with only 40 bytes of overhead per 1020 bytes of data. In my opinion, that's efficient. You might want to use this idea to save other types of data as part of your programs. It sure beats using data tapes.

The Slide Show is activated in subroutine 8000. This routine allows you to review the slides in any order, with whatever time delay you want between slides. You may even select a continuous Slide Show. (Note: This is useful for presentations. A computer store might use it for a promotion.) You can also make a ten-frame movie by simply setting the time between slides to zero. Line 8240 determines whether or not the slides continue. It is an INKEY routine that looks to see if the CLEAR key has been pressed. If the key is pressed, it finishes to the last slide, then exits the continuous routine.

One last note of caution regarding programming. It takes a while to type in all of those asterisks in the first 50 lines. If any of those lines has one too many or one too few characters, the program will self-destruct. Therefore, I suggest that you CSAVE the program before you RUN it. That way, if it has an error, all you have to do is CLOAD it, find your mistake, and fix it. One way to check your asterisks is to RUN the program and go immediately to the Slide Show. Wait for all ten slides to be printed. You should get ten identical screens full of asterisks with four bytes missing in the lower right corner.

A1	Number of slides to be viewed.
A2	Number of seconds between slides.
C	Equals 191. This is the graphics code for the block cursor used in the text mode.
CQ$	Contains answer to "Endless Slide Show" question.
L	Equals 15360. Address for start of CRT RAM.
PN	Slide number or print number.
TN	Tape number or tape letter.
U	Equals 16383. Address for end of CRT RAM.
V	Contains the location of the first byte of the slide to be printed. The equation X = USR(V) sends this address to the assembly program.
XM	Equals 17136. This is the address of the first asterisk or graphics byte in program line 1.
YM	Equal to slide number multiplied by 1060.
A, B, CQ, D, I, S, X, Y, Z	These are all temporary variables used for delay loops, counters, etc.

Table 1. *Variable list*

How it Works

When you run the program you'll get the list of the six key codes for going from one routine to the other. Once you've read the codes press the CLEAR key. You'll get a clear screen with an A in the lower right followed by a blinking block cursor. Enter a number from 0 to 9 and press ENTER. Now you can begin drawing. Press the space bar, and you'll see the blinking dot in the upper left corner. You can move the dot using the up, down, right, and left arrows. Moving the dot while pressing the space bar moves it without drawing. Releasing the space bar while moving the dot draws a line. To erase a dot or a line, simply go back over it while holding the space bar.

The A above and the number you type are tape codes. If you make several Slide Shows, you can permanently change the A to any other letter by typing another letter over it on any one of the slides while in the text mode. All the slides will be changed to the new letter. This is a handy reference. Slide C7, for example, would be slide 7 on tape C.

If you would like to add text to your slide, press the letter D followed by the letter T. The blinking block will show up again in the lower right of the screen. You can move it by using the arrows. The block is non-destructive. You can move it over the drawing without disturbing the drawing. To enter text, move the block into position and start typing. You can also change slide numbers and tape letters by typing over them with a new number or letter. To exit the text mode, press ENTER. Now you can add to or change the drawing, or enter a new command.

To record a slide to memory, press D followed by R. An R will appear in place of the slide number showing that it is recording. The recording takes approximately 25 seconds. When it's finished, the blinking block replaces the R and you are back in the text mode. If you fail to enter a slide number before recording to memory, it will record as slide zero for an ASCII character lower than 0, or as slide nine for an ASCII character of value greater than nine.

14590	Returns the raw code for the arrows and space bar, or combinations of these keys, as they are pressed.
15360	Start of CRT RAM.
16382	Location of tape number in CRT RAM.
16383	Location of slide number in CRT RAM. Lower right of CRT.
16526	Storage location for the LSB of the address for the start of the assembly program.
16527	Storage location for the MSB of the address for the start of the assembly program.
16537	Memory location where the TRS-80 stores the byte code of the last key pressed.
17136	RAM location for the first asterisk or graphics byte in program line 1.
27736	RAM location for storing the tape letter in program line 51.
31744–31775	Location of assembly routine in high memory.

Table 2. *Special address list*

To print any slide, enter the slide number in the lower right while in the text mode. Exit the text mode and press D followed by P. The slide will be instantly printed and you will again be in the text mode. You can wipe the screen clear by leaving the text mode and pressing D followed by W. When the screen clears you'll be in text mode. To remove a slide from memory, enter the slide number in the lower right, and exit the text mode. Press D followed by C. The letter C will appear where the slide number was and remain until the slide is cleared (about 15 to 20 seconds). The text mode will automatically return. It is necessary to clear all slides if you wish to list the program. The slides saved in the REM statements will not list properly.

For the Slide Show, exit the text mode and press D followed by S. The screen will clear, and you'll be asked how many slides you wish to see and in what order. Enter the number for the first slide you want. Question marks will continue coming until you've entered as many slide numbers as you'd originally asked to see. The next question is for the delay, in seconds, that you want between slides. At this point, you could transform your slide program into an audiovisual show. With a little program change the TRS-80 could turn on and off a tape recorder to accompany the slides. The last question is for the continuous slide show. Answer Yes or No or Y or N. If you enter Yes, it will show the slides in continuous order. To break the loop, press the CLEAR key. The loop will continue to the last slide and then stop back in the text mode. (Too bad there is no easy way to get the computer to rewind the tape in the tape recorder. If it could, you could have an automatic audiovisual presentation.)

Program Listing. *Slide Show*

Encyclopedia Loader

```
1  '*************************************************************
****************************************************************
****************************************************************
***************
2  '*************************************************************
****************************************************************
****************************************************************
***************
3  '*************************************************************
****************************************************************
****************************************************************
***************
4  '*************************************************************
****************************************************************
****************************************************************
***************
5  '*************************************************************
****************************************************************
****************************************************************
***************
6  '*************************************************************
****************************************************************
****************************************************************
***************
7  '*************************************************************
****************************************************************
****************************************************************
***************
8  '*************************************************************
****************************************************************
****************************************************************
***************
9  '*************************************************************
****************************************************************
****************************************************************
***************
10 '************************************************************
****************************************************************
****************************************************************
****************
11 '************************************************************
****************************************************************
****************************************************************
****************
12 '************************************************************
****************************************************************
****************************************************************
****************
13 '************************************************************
****************************************************************
****************************************************************
****************
14 '************************************************************
****************************************************************
****************************************************************
****************
15 '************************************************************
****************************************************************
****************************************************************
****************
16 '************************************************************
****************************************************************
****************************************************************
****************
17 '************************************************************
****************************************************************
****************************************************************
```

```
****************
18 '************************************************************
****************************************************************
****************************************************************
****************
19 '************************************************************
****************************************************************
****************************************************************
****************
20 '************************************************************
****************************************************************
****************************************************************
****************
21 '************************************************************
****************************************************************
****************************************************************
****************
22 '************************************************************
****************************************************************
****************************************************************
****************
23 '************************************************************
****************************************************************
****************************************************************
****************
24 '************************************************************
****************************************************************
****************************************************************
****************
25 '************************************************************
****************************************************************
****************************************************************
****************
26 '************************************************************
****************************************************************
****************************************************************
****************
27 '************************************************************
****************************************************************
****************************************************************
****************
28 '************************************************************
****************************************************************
****************************************************************
****************
29 '************************************************************
****************************************************************
****************************************************************
****************
30 '************************************************************
****************************************************************
****************************************************************
****************
31 '************************************************************
****************************************************************
****************************************************************
****************
32 '************************************************************
****************************************************************
****************************************************************
****************
33 '************************************************************
****************************************************************
****************************************************************
****************
34 '************************************************************
****************************************************************
****************************************************************
****************
35 '************************************************************
```

Program continued

```
********************************************************************
********************************************************************
****************
36 '****************************************************************
********************************************************************
********************************************************************
****************
37 '****************************************************************
********************************************************************
********************************************************************
****************
38 '****************************************************************
********************************************************************
********************************************************************
****************
39 '****************************************************************
********************************************************************
********************************************************************
****************
40 '****************************************************************
********************************************************************
********************************************************************
****************
41 '****************************************************************
********************************************************************
********************************************************************
****************
42 '****************************************************************
********************************************************************
********************************************************************
****************
43 '****************************************************************
********************************************************************
********************************************************************
****************
44 '****************************************************************
********************************************************************
********************************************************************
****************
45 '****************************************************************
********************************************************************
********************************************************************
****************
46 '****************************************************************
********************************************************************
********************************************************************
****************
47 '****************************************************************
********************************************************************
********************************************************************
****************
48 '****************************************************************
********************************************************************
********************************************************************
****************
49 '****************************************************************
********************************************************************
********************************************************************
****************
50 '****************************************************************
********************************************************************
********************************************************************
****************
51 'A
1000 :
       '  ******* INITIALIZE SYSTEM *******
1020 DEFINT B - Z:
     DEFSTR S:
     L  = 15360:
     XM  = 17136:
```

```
     U = 16383:
     C = 191:
     CLS
1040 POKE 16526,0:
     POKE 16527,124
1060 DATA 205,127,10,213,197,245,62,5,17,0,60,1,204,0,237,176,6,8,35,
     16,253,61,194,11,124,241,193,209,195,154,10,0
1080 FOR X = 31744 TO 31775 :
      READ Y :
      POKE X,Y :
      NEXT X
1100 PRINT "THE CODES TO OPERATE THIS PROGRAM ARE:"
1120 PRINT "1)DT---THIS ENTER'S THE TEXT MODE. TO EXIT PRESS 'ENTER'"
1140 PRINT "2)DR---THIS STORES THE SLIDE IN MEMORY AS THE SLIDE NUMBE
     R              IN THE LOWER RIGHT"
1160 PRINT "3)DP---THIS PRINTS THE SLIDE AS NUMBERED IN THE LOWER RIG
     HT."
1180 PRINT "4)DW---THIS IS A 'CLEAR SCREEN'"
1200 PRINT "5)DC---CLEARS THE SLIDE AS NUMBERED IN THE LOWER RIGHT FR
     OM MEMORY"
1220 PRINT "6)DS---THIS ENTERS INTO THE SLIDE SHOW ROUTINE.":
     PRINT "        NOTE:TO EXIT A CONTINOUS SLIDE SHOW, PRESS 'CLEAR'
     ."
1240 PRINT :
     PRINT :
     PRINT "PRESS 'CLEAR' TO START"
1260 IF PEEK(16537) = 31
      THEN
       CLS :
       POKE 16382, PEEK(27736):
       GOTO 4000:
      ELSE
       1260
2000 :
     ' ******* DRAWING *******
2020 X = 0:
     Y = 0
2040 IF INKEY$ = "D"
      THEN
       3020
2060 B = PEEK(14590):
     IF B = 0 OR B = 24 OR B = 96
      THEN
       2040:
      ELSE
       IF B = 128
        THEN
         2160
2080 IF B = 8 OR B = 40 OR B = 72 OR B = 136 OR B = 168 OR B
     = 200Y = Y - 1:
     IF Y < 0Y = 0
2100 IF B = 16 OR B = 48 OR B = 80 OR B = 144 OR B = 176 OR B
     = 208Y = Y + 1:
     IF Y > 47Y = 47
2120 IF B = 32 OR B = 40 OR B = 48 OR B = 160 OR B = 168 OR B
     = 176X = X - 1:
     IF X < 0X = 0
2140 IF B = 64 OR B = 72 OR B = 80 OR B = 192 OR B = 200 OR B
     = 208X = X + 1:
     IF X > 127X = 127
2160 SET(X,Y)
2180 IF B > 85 FOR I = 1 TO 10:
      NEXT :
     RESET(X,Y)
2200 GOTO 2040
3000 :
     ' ******* SELECTS WHICH SUBROUTINE *******
3020 S = INKEY$:
     IF S = ""
      THEN
```

Program continued

```
      3020
3040 PN = PEEK(16383) - 48
3060 IF PN < 0
      THEN
       PN = 0 :
      ELSE
       IF PN > 9
        THEN
         PN = 9
3080 TN = PEEK(16382):
     POKE 27736,TN
3100 YM = PN * 1060
3120 IF S = "T"
      THEN
       4000
3140 IF S = "C"
      THEN
       5000
3160 IF S = "R"
      THEN
       6000
3180 IF S = "P"
      THEN
       7000
3200 IF S = "S"
      THEN
       8000
3220 IF S = "W"
      THEN
       CLS :
       POKE 16382, PEEK(27736)
3240 GOTO 4000
4000 :
     ' ******* TEXT *******
4020 A = U
4040 D = PEEK(A):
     POKE A,C
4060 S = INKEY$:
     IF S = "" POKE A,32:
     FOR I = 1 TO 20:
      NEXT :
     POKE A,D:
     GOTO 4040
4080 B = ASC(S)
4100 IF B = 13 POKE A,D:
     GOTO 2020
4120 IF B > 31 AND B < 91 POKE A,B:
     A = A + 1:
     IF A > U
      THEN
       A = U
4140 IF B = 8 POKE A,D:
     A = A - 1:
     IF A < L
      THEN
       A = L
4160 IF B = 9 POKE A,D:
     A = A + 1:
     IF A > U
      THEN
       A = U
4180 IF B = 10 POKE A,D:
     A = A + 64:
     IF A > U
      THEN
       A = A - 64
4200 IF B = 91 POKE A,D:
     A = A - 64:
     IF A < L
      THEN
```

```
       A = A + 64
4220 GOTO 4040
5000 :
     ' ******* CLEARS SLIDE MEMORY ONE SLIDE AT A TIME *******
5020 POKE 16383,67
5040 FOR Y = 1 TO 5:
      FOR X = 0 TO 203
5060   POKE XM + YM + X,42
5080   NEXT X
5100  YM = YM + 212:
      NEXT Y
5120 POKE 16383,PN + 48
5140 GOTO 4000
6000 :
     ' ******* RECORDS SLIDE TO MEMORY *******
6020 POKE 16383,82
6040 FOR Y = 0 TO 816 STEP 204:
      FOR X = 0 TO 203
6060   Z = PEEK(15360 + X + Y)
6080   POKE XM + YM + X,Z
6100   NEXT X
6120  YM = YM + 212:
      NEXT Y
6140 POKE 16383,PN + 48
6160 GOTO 4000
7000 :
     ' ******* REDRAWS SLIDE *******
7020 POKE 16382,TN
7040 V = XM + YM:
     X = USR(V)
7060 POKE 16383,PN + 48
7080 IF CQ < > 0
      THEN
       RETURN :
      ELSE
       4000
8000 :
     ' ******* SLIDE SHOW *******
8020 CLS :
     INPUT "HOW MANY SLIDES WOULD YOU LIKE TO VIEW";A1
8040 PRINT "IN WHAT ORDER WOULD YOU LIKE TO VIEW THEM"
8060 FOR CQ = 1 TO A1:
      INPUT A1(CQ)
8080  IF A1(CQ) < 0 OR A1(CQ) > 9
       THEN
        A1(CQ) = 0
8100  NEXT CQ
8120 INPUT "HOW MANY SECONDS BETWEEN SLIDES";A2
8140 INPUT "WOULD YOU LIKE AN ENDLESS SHOW";CQ$
8160 FOR CQ = 1 TO A1:
      PN = A1(CQ):
      YM = PN * 1060:
      GOSUB 7000
8180  FOR DELAY = 1 TO 333 * A2:
       NEXT DELAY
8200  NEXT CQ
8220 CQ = 0
8240 IF PEEK(16537) = 31
      THEN
       GOTO 4000
8260 IF LEFT$(CQ$,1) = "Y"
      THEN
       8160
8280 GOTO 4000
```

Graphs, Plus

by Allan S. Joffe W3KBM

The plus in "Graphs, Plus" means that the routines contained in the programs are useful in your own cookbook. I use the term "graph" to indicate a plotted curve that has a relationship to the data points used to develop the curve.

How the Program Works

Program Listing 1 plots the graph of a sine wave (see Photo 1). Lines 3 through 30 clear the screen (CLS) and set aside string space for setting the background, which appears first. Lines 50, 55, and 60 input the data values of the sine wave to be graphed. In line 55, the * 15 at the end of the line sets the amplitude of the sine wave. Line 60 centers the sine wave on the screen.

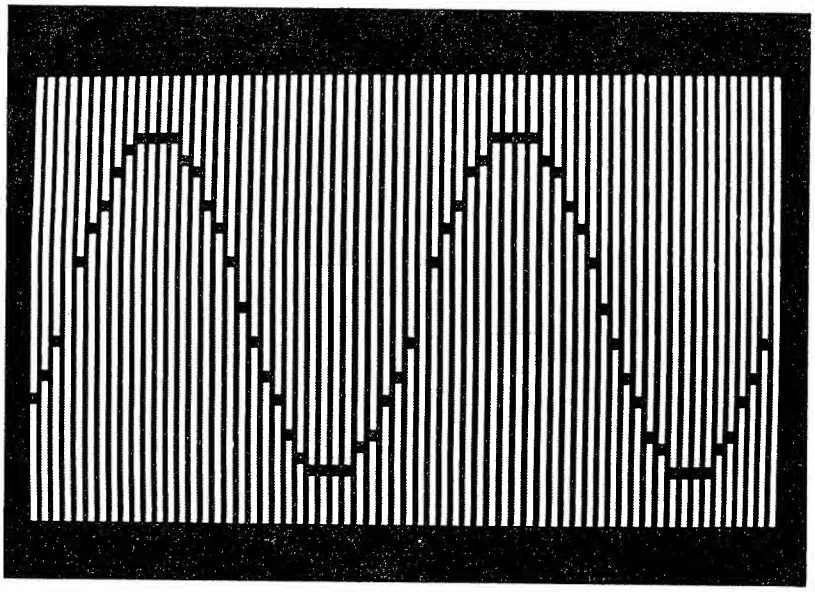


Photo 1. *Sine wave plot with background (Photos by Ira Joffe)*

In line 55, the expression SIN(X*6) sets the number of complete sine waves that will appear. If you change this to SIN(X*3), then one wave is graphed; if it appeared as SIN(X*12), four waves would be graphed. As pointed out in the remark in line 45, a CLS and RUN 50 dispenses with the background and prints out the graph as a series of properly placed spots of light.

If you eliminate the STEP 2 in line 50, the graph assumes a "white on white" character. This is a function of the POINT and NOT POINT statements in lines 70 and 75. If you would like to see a "whiter on white" effect at this point, delete line 70.

A different type of graphing plot is a modulation envelope type of graph (see Photos 2 and 3). Program Listing 2 gives the TRS-80 instructions for this display. The program up to line 40 calculates the data points that form the curve. Line 30 sets the number of waveforms to be plotted and controls the amplitude of the image. Lines 50 and 60 center the image on the monitor screen. The balance of the program sets the points of light that produce the image.

If you follow the suggestion in the remark in line 90, you will see the equivalent of half-wave rectification. If you then delete this insertion, you can change line 50 to read:

```
Y = 23 + (- J)
```

When you are in the mood to think a bit, try to analyze why the new line 50 does what it does.

Bar Graphs

Program Listing 3 gives your computer the data and directions it needs to implement a vertical bar graph (see Photo 4). Lines 20 through 40 input random data so that you do not have to type in 16 numbers to see the program run. If you want to input your own figures at a later date, then change line 30 to:

```
INPUT P(R)
```

Notice that, since you are dealing with the graph on an elemental level (which precludes too much on scaling factors), if any P(R) value exceeds 47, then you bump into the top of the screen and generate an error message. This makes the random input statement convenient for seeing how the program runs; the intent here is on generating the graph more than anything else.

If you watch the program run, you will see that a point is set at the maximum value of each data element, and then the graph line is finished. The routine in line 70 sets this point. The remainder of the program is fairly easy to grasp.

Now if you add the program modification shown in Program Listing 4 to Program Listing 3, you can produce the same graph in a reverse video presentation (see Photo 5). In lines 70 and 80, change SET to RESET. This completes the changes for the reverse video presentation of the graph.

Before you go on to the horizontal bar graph, consider that the fundamental vertical bar graph has a long, horizontal base line of 128 positions, so that if we used every fourth one, we could plot 32 independent pieces of data with some viewing comfort. It has a potential maximum (without scaling) of 48 in the vertical dimension.

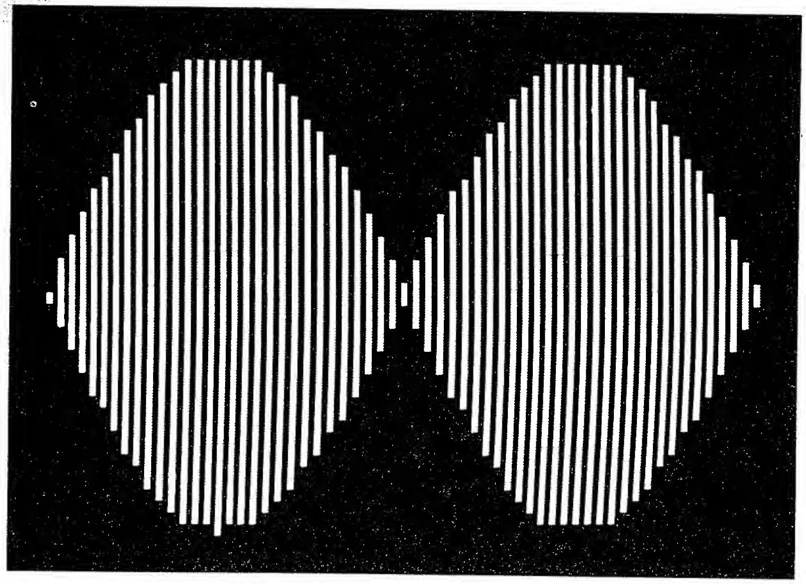


Photo 2. *Modulation envelope type of display*

If you turn the vertical graph on its side, these two factors are reversed. It may seem that you haven't gained anything, but you have. Your data values can now go as high as 128. Holding them to a maximum value of 100 simplifies scaling, if and when needed. If you are graphing data to be expressed as a percentage, then this type of graph is a natural.

Program Listing 5 provides one way to program the horizontal bar graph. Notice that the RND function in line 30 is RND(127), so that when you run the demonstration program, you stand the chance of using all 128 horizontal positions available in the TRS-80 graphics. The program is concise and to the point. The expression in line 80, (T,3*X – 2), controls the vertical separation of the adjacent horizontal graph bars.

In order to make room for the printing of values, the maximum data value for any one bar ranges from 0 to 100; so for the demonstration program, line 30 shows a RND value of RND (100). Lines 45, 55, 60, and 80 print data values. In line 60, if CHR$(61) is used, the bars are formed of equal signs (=); CHR$(45) in line 60 produces a dash (—) graph; CHR$(46) produces a period (.) graph; and CHR$(58) produces colons (:). My personal preference is either the equal sign or the colon. The faster printing of the graph is due to the construction of line 60, which uses the PRINT command rather than the SET command.

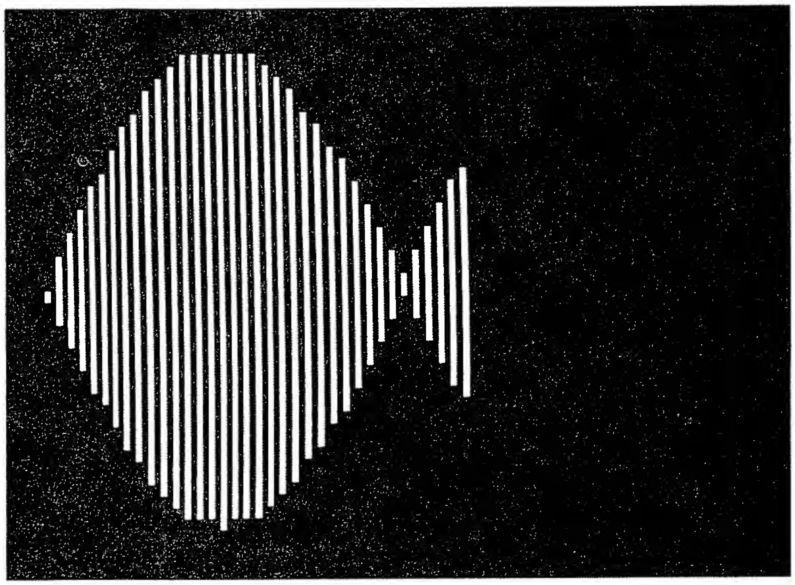


Photo 3. *Fancy fish using modulation envelope display*

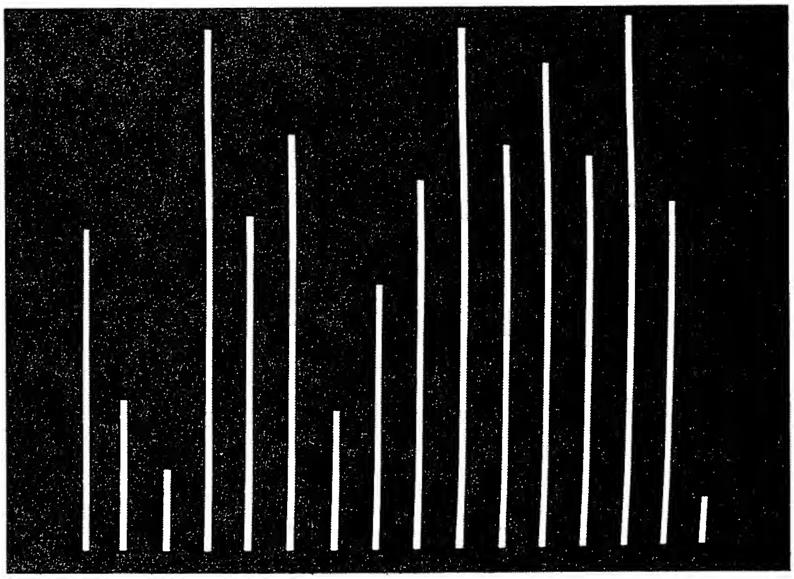


Photo 4. *Vertical bar graph—normal video display*

Program Listing 6 continues the evolution of the horizontal bar graph with items added to the fundamental program shown in Program Listing 5 (see Photo 6). First of all, Program Listing 6 offers a graph that prints out on the screen in a much shorter time than the graph generated by Program Listing 5. Next each bar of the graph is supplied with annotation; a printed value is shown at each bar position to indicate the value of the bar. Finally,

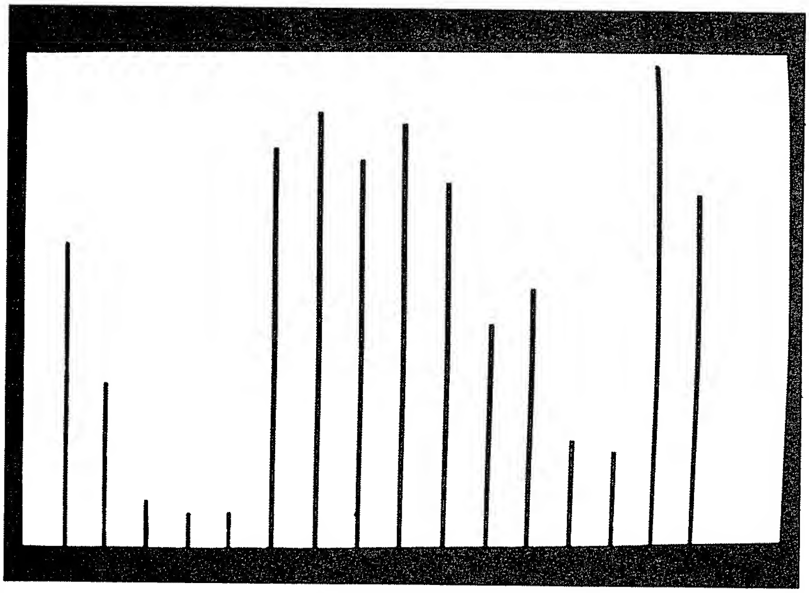


Photo 5. *Vertical bar graph—reverse video display*

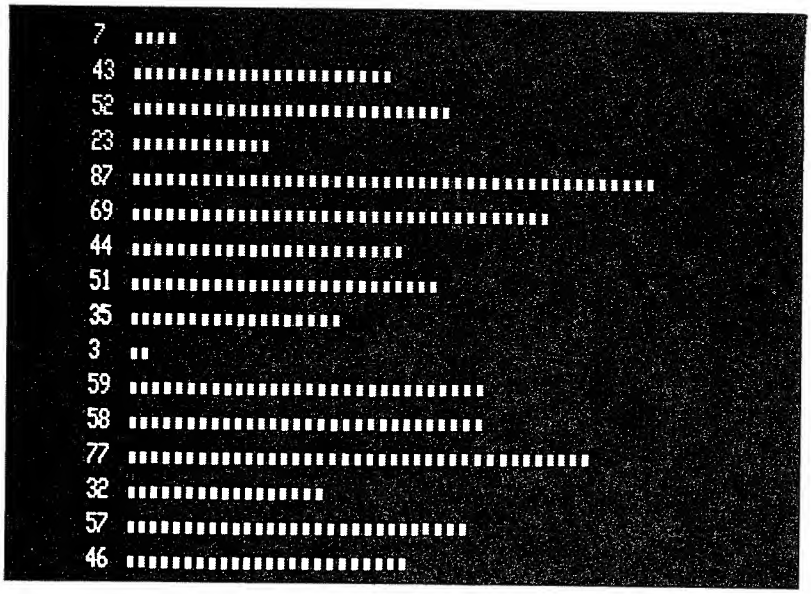


Photo 6. *Annotated horizontal bar graph*

Program Listing 6 includes the option of selecting the print form of the bar itself. This is a question of taste; so once the method is made clear, you can please yourself.

Program Listing 1. *(In line 20, [a] is the @ symbol)*

```
 3 CLS
 5 REM  GRAPH OF SINCE WAVE WITH OR WITHOUT BACKGROUND
 7 CLEAR 150
10 FOR X = 0 TO 767 STEP 48
20  PRINT @ X, STRING$(112, CHR$(149))
30  NEXT X
40 REM  LINES TO THIS POINT SET BACKGROUND
45 REM  CLS AND RUN 50 GIVES SINE CURVE WITHOUT BACKGROUND
50 FOR X = 0 TO 120 STEP 2
55  J = SIN((X * 6) * .01745) * 15
60  Y = 19:
    REM  VERTICAL CENTERING FACTOR
70  IF POINT (X,Y) RESET (X,Y - J)
75  IF NOT POINT (X,Y) SET (X,Y - J)
80  NEXT X
90 END
```

Program Listing 2

```
  5 REM  MODULATION ENVELOPE BAR GRAPH DISPLAY
 10 CLS
 20 FOR X = 0 TO 120 STEP 2
 30  J = SIN((X * 3) * .01745) * 20
 40  IF J < 0
      THEN
       J = ABS(J)
 45  REM  NEXT TWO LINES CONTROL VERTICAL CENTERING
 50  Y = 23 + J
 60  W = 23 - J
 70  FOR V = W TO 23
 80   SET (X,V)
 90   NEXT V:
     REM  TRY ADDING LINE 95 GOTO 140 -- IT'S RECTIFYING
110  FOR V2 = 23 TO Y
120   SET (X,V2)
130   NEXT V2
140  NEXT X
150 GOTO 150
```

Program Listing 3

```
  7 REM  VERTICAL BAR GRAPH
 10 DIM P(20)
 15 CLS
 20 FOR R = 1 TO 16
 30  P(R) = RND(45)
 40  NEXT R
 50 FOR X = 1 TO 16
 60  L = 45 - P(X)
 70  SET(7 * X,(45 - P(X)))
 80  FOR T = 45 TO L STEP - 1:
      SET (7 * X,T):
      NEXT T
 90  NEXT X
100 GOTO 100
```

Program Listing 4. *Reverse video modification*

```
   5 CLEAR 500
  47 GOSUB 2000
2000 C$ = STRING$(192, CHR$(191)):
     PRINT C$;C$;C$;C$;
2010 PRINT STRING$(128, CHR$(191));
2020 RETURN
```

Program Listing 5

```
  1 REM  HORIZONTAL BAR GRAPH
  5 CLS
 10 DIM P(20)
 20 FOR R = 1 TO 16
 30  P(R) = RND(127)
 40  NEXT R
 50 FOR X = 1 TO 16
 60  L = P(X)
 80  FOR T = 0 TO L STEP 2:
      SET (T,3 * X - 2):
      NEXT T
 90  NEXT X
100 GOTO 100
```

Program Listing 6

```
  1 REM  ANNOTATED HORIZONTAL BAR GRAPH
  5 CLS
 10 DIM P(100)
 20 FOR R = 1 TO 16
 30  P(R) = RND(100)
 40  NEXT R
 45 Q = 0
 50 FOR X = 1 TO 16
 55  PRINT @ Q, P(X);
 60  PRINT @ Q + 5, STRING$((P(X) / 2), CHR$(61));
 80  Q = Q + 64
 90  NEXT X
100 GOTO 100
```

HARDWARE

Interrupt Mode 1.5
Reverse Video Hardware Modification

Interrupt Mode 1.5

by David Haan

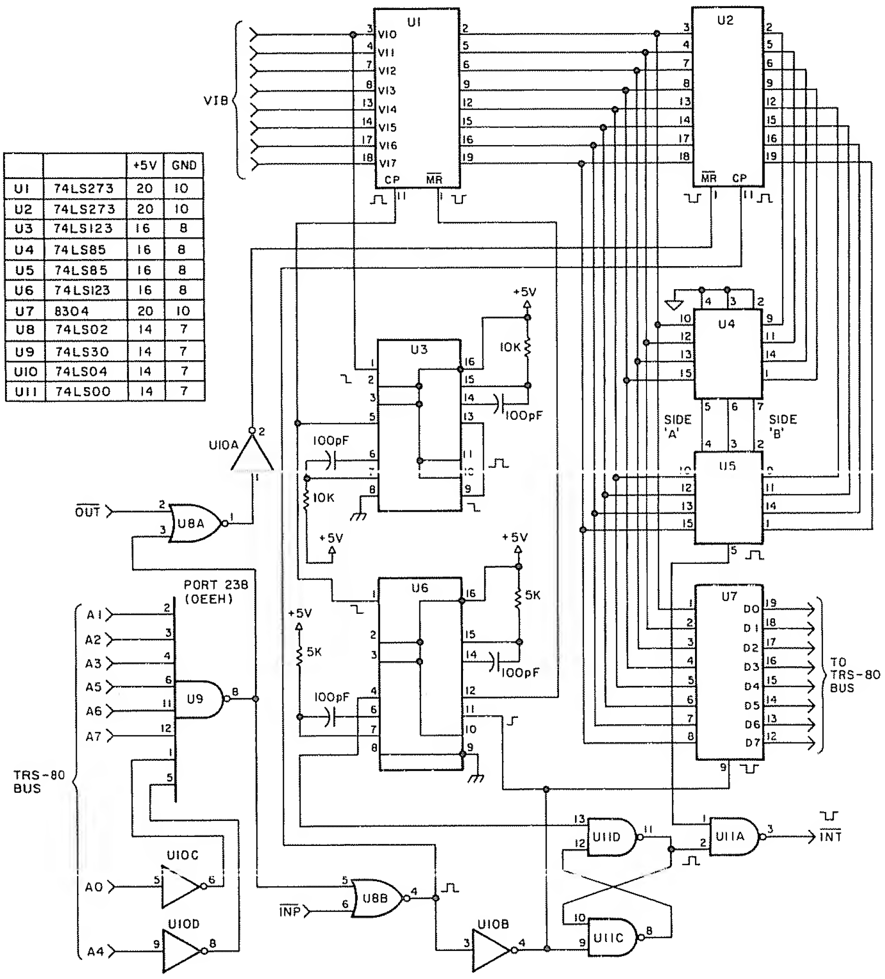
This article deals with the interrupts of the TRS-80—specifically, getting the mode 1 interrupt to operate in a manner similar to that of mode 2.

The Z-80 provides power for interrupt processing, particularly for mode 2. The TRS-80 as it stands, however, does not support mode 2 because the IORQ and M1 signals are not available, and Z-80 support chips such as PIAs and SIAs require these signals under mode 2. This leaves us with mode 1, but the advantages of mode 2 are hard to overlook. Mode 2 is faster than any other mode and can call up to 128 interrupt service routines directly, anywhere in memory. Mode 1, on the other hand, can call only one routine which must be at 38H. The TRS-80 uses mode 1, and when interrupted, looks at a hardware address in the expansion interface. By determining which bit is set in the byte that is returned, it can determine whether the clock or disk controller interrupted the CPU. It then executes the appropriate routine.

You can take this process one step further and have the TRS-80 look at an interrupt-handling board and determine which of up to 128 outside devices is interrupting the CPU. A board of this sort is shown in Figure 1. Since we are using mode 1, the IORQ and M1 signals aren't used. We will need a little software to go with the board to simulate mode 2. This scheme requires about 80 microseconds to access the correct interrupt service routine (ISR) and begin execution.

The board receives the interrupting device's priority level and determines if this interrupt is of a higher priority than the current ISR, in execution. If it is higher, the board generates a new interrupt. Upon completion of the last ISR, the board is cleared and made ready for a new series of interrupts.

To see how this process works, let's follow a manual interrupt and see what hardware and software considerations are necessary. We will cover the software first. Our interrupt priority levels must be between 0 and 254 in multiples of 2. This is because the board forms an index into a table of 16 bit addresses. We will arbitrarily set the manual-interrupt priority level to 0FEH (254), the highest level we can have. Next, we must set up the table which contains the address pointers of all the ISRs. This same step would be required if we were operating under mode 2. We will set up our table to start at 7F00H. Since each of the interrupting devices needs two bytes for its ISR address, the table will be 128 × 2 bytes, or 256 bytes long. As an example, if a device has a priority level of 50H, its ISR address would be located at



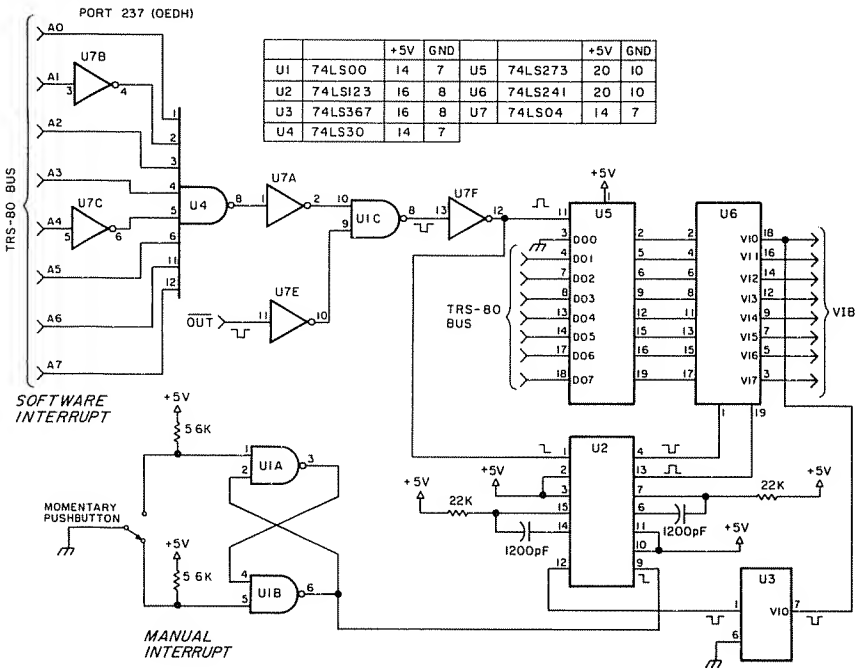
		+5V	GND
U1	74LS273	20	10
U2	74LS273	20	10
U3	74LS123	16	8
U4	74LS85	16	8
U5	74LS85	16	8
U6	74LS123	16	8
U7	8304	20	10
U8	74LS02	14	7
U9	74LS30	14	7
U10	74LS04	14	7
U11	74LS00	14	7

Figure 1. *Vector interrupt-handling board*

7F50H and 7F51H of our table. Our manual interrupt has a priority level of 0FEH, and I have set its ISR to reside at 7D00H; so, 00H will be loaded into location 7FFEH, and 7DH will be loaded into location 7FFFH. (The low-order byte goes in the low-order memory location.) The loading of the manual interrupt ISR address pointer is shown in Section 1 of the Program Listing. Next, we need to tell the CPU where the table of ISR address pointers is located. To do this in mode 2, we would put the high-order byte at the start of our table in the I register, or I = 7FH. The interrupting device would provide the low-order byte to form the address of the pointer which points to the address of the appropriate ISR. Since we can't do this in mode 1, we have to do it through an interrupt processor. This interrupt processor is shown in Section 3 of the Program Listing. When an interrupt is received, the CPU does a call to 38H where a jump to 4012H is located. At 4012H, we put in another jump instruction to cause the CPU to jump to our interrupt processor. This is done by executing the code of Section 2 of the Program Listing.

The interrupt processor now gets the priority level from the interrupt-handling board, gets the appropriate ISR address from the table, and does a call to it. Once the CPU knows where our table is and how to get to our ISR, we need to provide the ISR of our manual interrupt. This is shown in Section 4 of the Program Listing which saves the CRT image, clears the CRT, and puts up a message. Before returning to BASIC, the routine restores the CRT. Since we have all the software we need, let's mentally push the manual interrupt button and examine the operation of the hardware.

A schematic for the manual interrupt circuit is shown in Figure 2. The one-shot U2 turns on the tri-state buffer, which puts the priority level (0FEH) on a common bus I call the vector interrupt bus (VIB). The one-shot and tri-state buffers are required on each device that interrupts the CPU, since each puts its priority-level byte on the VIB in parallel. It can be there only long enough to trigger U3 on the interrupt-handling board—thus the name one-shot. Figure 2 also shows a software interrupt circuit that can generate the interrupt priority level you choose.



		+5V	GND			+5V	GND
U1	74LS00	14	7	U5	74LS273	20	10
U2	74LS123	16	8	U6	74LS241	20	10
U3	74LS367	16	8	U7	74LS04	14	7
U4	74LS30	14	7				

Figure 2. *Manual and software interrupts*

At this point, U1 on the interrupt-handling board receives the priority level on the VIB. As line V10 goes low (it must always go low since our priority levels are multiples of 2), it will trigger U3 which generates a clock pulse to U1 and U6 after a short delay. This delay gives the data coming to U1 time to settle down. U1 then distributes the priority level to U2, U4, U5, and U7. After U6 is triggered by U3, it triggers U11, which toggles pin 11 of U11. This sets up an interrupt to the CPU, but the interrupt is not sent to the CPU at this time.

U4 and U5 serve as an eight-bit word comparator. In this case, the priority level of 0FEH on side A of U4 and U5 is compared with the priority level on side B, which is initially zero. If side A is of higher priority than side B (which it is), pin 5 of U5 is set high allowing pin 3 of U11 to issue the interrupt to the CPU.

U7 is a tri-state driver which sends the priority level when the CPU requests it. Once the CPU has an interrupt, it calls location 38H, jumps to 4012H, then jumps to our interrupt processor. The processor loads the A register with the priority level from U7. It uses this value as an offset into the table of address pointers to the ISRs. The address of the appropriate ISR is fetched, placed into a call, and then the call is executed. At this point, you are in the ISR for the manual interrupt.

A few things caused by the execution of our interrupt processor routine were going on in the interrupt-handling board. When the CPU went to U7 to get the priority level, U2 was clocked by the leading edge of the signal from pin 4 of U8. This caused the priority level to be loaded into U2, which in turn output 0FEH to side B of U4 and U5. Since side B had the priority level of 0FEH, and side A had 0FEH, pin 5 of U5 went low, turning off the interrupt to the CPU. U6 was then clocked by the trailing edge of the pulse from pin 4 of U10, and U6 sent a reset to U1. As a result, side A of U4 and U5 now has a zero.

At this point, anything could be in our ISR; Section 4 of the Program Listing is intended for demonstration only. By hitting ENTER, you return to the interrupt processor and check the interrupt-handling board to see if any lower priority level interrupt has occured. If not, the OUT (0EFH),A clears the board, and you return to BASIC. If another interrupt of lower priority than the one you just executed is waiting, the interrupt processor is reentered, and the new ISR is executed. If another interrupt of higher priority is received during an ISR, the interrupt-handling board interrupts the CPU and the higher priority level interrupt is serviced. If each interrupt received is successively higher priority, from 2 to 254, you could stack up 127 ISRs, assuming none had a chance to complete before it was interrupted. Only one lower priority level interrupt can be saved, however, and only the last one received will be executed, because each interrupt is overwritten by the next if the previous one has not yet been serviced. I said earlier that there are 128 ISRs which can be used. The interrupt-handling board uses ISR 0 for comparing all the other ISRs; so the lowest ISR priority level available is level 2. For this reason, we can stack up only 127 ISRs.

To ensure that you return from one ISR to a lower priority ISR which was interrupted, the code in lines 790 through 840 must be the first code executed in each of your ISRs. The code in lines 1120 to 1170 must be the last code executed in each ISR. This establishes a return instruction at lines 560 and 570 of the interrupt processor of Section 3 of the Program Listing.

To see Interrupt Mode 1.5 work, build the circuits shown in Figures 1 and 2. I wire wrapped mine to simplify testing and modification. Because of the power draw, you'll need a separate 5-volt power supply. After connecting the interrupt-handling board to the TRS-80 expansion edge connector using an appropriate connector and ribbon cable, set memory size to 30700, load all the software in Sections 1 through 4 of the Program Listing, and execute the software in Section 1. This returns you to BASIC. Load any BASIC program that doesn't POKE into the area above 30700 and which has some display on the screen and run it. While it is running, hit the manual interrupt button, and the screen should display a message. After hitting the ENTER key, you should return to your BASIC program with the CRT intact and ready to continue execution of your BASIC program.

The software you use for your ISRs depends entirely on your imagination and on what you have interrupting the CPU. A clock routine which requires interrupts similar to the one Radio Shack uses can be used, and now you can prioritize its execution. Any other interrupt can also be prioritized so that the more important one is serviced immediately when the device interrupts, while blocking out lower priority level interrupts.

I use the manual interrupt along with other interrupting devices on an S-100 bus I have interfaced to the TRS-80, including an A/D and a software interrupt circuit. The software interrupt circuit shown in Figure 2 is an I/O port which, when I output a byte to it (in multiples of 2), generates an interrupt at whatever priority level I sent. For instance, if I send out a byte of 1CH, an interrupt of priority level 28 is generated. In this way, I can access any ISR or machine-language program directly from any BASIC or machine-language program. In other words, I have 127 routines I can access, if necessary, instead of only the one to 10 USR functions Radio Shack provides.

A word of caution: Since this interrupt-handling board uses the INT line on the TRS-80, it would preclude the use of the expansion interface, because the clock and disk controller use the INT line. Neither the gate of the interrupt-handling board nor the gate used on the expansion interface is a tri-state or open-collector device. As such, the interrupt-handling board would pull the TTL output gate in the expansion interface low.

Program Listing. *Software for the Interrupt Service Routine*

Encyclopedia Loader

```
                 ;                      SECTION 1
                 ;
                 ;    THIS ROUTINE WILL PLACE THE MANUAL INTERRUPT
                 ;    ISR ADDRESS INTO THE TABLE OF ISR'S.
                 ;
7DØØ                    ORG     7DØØH
                 ;
7DØØ 21927E      START  LD      HL,MANINT          ;MAN. INT. ISR LOCATION.
7DØ3 22FE7F             LD      (7FFEH),HL         ;PLACE IN ISR TABLE.
                 ;
                 ;
                 ;                     SECTION 2
                 ;
                 ;    THIS IS THE SET UP FOR THE INTERRUPT PROCESSOR.
                 ;    IT MUST BE RUN TO ESTABLISH THE LOCATION OF THE
                 ;    INTERRUPT PROCESSOR.  IT NEED NOT BE IN PROTECTED
                 ;    MEMORY.
                 ;
7DØ6 F3                 DI                         ;DISABLE INTERRUPTS.
7DØ7 DBEE               IN      A,(ØEEH)           ;THE IN AND OUT  IN-
7DØ9 D3EE               OUT     (ØEEH),A           ;STRUCTIONS  CLEAR  THE
                                                   ;INTERRUPT BOARD.
7DØB 3EC3               LD      A,ØC3H             ;LOAD JUMP INSTRUCTION.
7DØD 32124Ø             LD      (4Ø12H),A          ;INTO MEM. LOC. 4Ø12H.
7D1Ø 21ØØ7E             LD      HL,IPROC           ;START OF INT. PROCESSOR.
7D13 22134Ø             LD      (4Ø13H),HL         ;PUT IN JUMP ADDRESS.
7D16 ED56               IM      1                  ;SET INT. MODE TO 1.
7D18 FB                 EI                         ;ENABLE INTERRUPTS.
7D19 C3191A             JP      1A19H              ;GO TO BASIC ENTRY.
                 ;
                 ;
                 ;                     SECTION 3
                 ;
                 ;    THIS IS THE INTERRUPT PROCESSOR.  IT MUST
                 ;    RESIDE IN PROTECTED MEMORY.
                 ;
7EØØ                    ORG     7EØØH
                 ;
7EØØ F5          IPROC  PUSH    AF                 ;SAVE AF REGISTERS.
7EØ1 C5                 PUSH    BC                 ;SAVE BC REGISTERS.
7EØ2 DBEE               IN      A,(ØEEH)           ;GET INT. PRIORITY LEVEL.
7EØ4 4F          IPROC1 LD      C,A                ;REG. 'C' HAS OFFSET
                                                   ;INTO ISR TABLE.
7EØ5 Ø67F               LD      B,7FH              ;REG. 'B' HAS START  OF
                                                   ;ISR TABLE.
7EØ7 ØA                 LD      A,(BC)             ;ISR LO ORDER BYTE.
7EØ8 32137E             LD      (CALL1),A          ;PUT ADDR. IN CALL INSTR.
7EØB ØC                 INC     C
7EØC ØA                 LD      A,(BC)             ;ISR HI ORDER BYTE.
7EØD 32147E             LD      (CALL2),A          ;PUT ADDR. IN CALL INSTR.
7E1Ø C1                 POP     BC                 ;RESTORE 'BC' REGISTERS.
7E11 F1                 POP     AF                 ;RESTORE 'AF' REGISTERS.
7E12 CD          CALL   DEFB    ØCDH               ;CALL INSTRUCTION.
ØØØ1             CALL1  DEFS    1                  ;CALL LO ORDER BYTE.
ØØØ1             CALL2  DEFS    1                  ;CALL HI ORDER BYTE.
7E15 F5          CNTRL  PUSH    AF                 ;SAVE 'AF' REGISTERS.
7E16 C5                 PUSH    BC                 ;SAVE 'BC' REGISTERS.
7E17 DBEE               IN      A,(ØEEH)           ;GET INT. PRIORITY LEVEL.
7E19 FEØØ               CP      Ø                  ;TEST IF ONE EXITS.
7E1B 2ØE7               JR      NZ,IPROC1          ;GET ISR ADDRESS.
7E1D D3EE               OUT     (ØEEH),A           ;CLEAR INT. BOARD.
7E1F C1                 POP     BC                 ;RESTORE 'BC' REGISTERS.
7E2Ø F1                 POP     AF                 ;RESTORE 'AF' REGISTERS.
7E21 C9                 RET                        ;RETURN FROM INTERRUPT.
                 ;
                 ;
                 ;                      SECTION 4
```

```
               ØØ68Ø ;
               ØØ69Ø ;     THIS IS THE MANUAL INTERRUPT SERVICE ROUTINE.
               ØØ7ØØ ;     IT MUST ALSO RESIDE IN PROTECTED MEMORY.
               ØØ71Ø ;
7E4Ø           ØØ72Ø           ORG     7E4ØH
               ØØ73Ø ;
7E4Ø 59        ØØ74Ø MESS1     DEFM    'YOU CAN PLACE ANYTHING YOU '
7E5B 57        ØØ75Ø           DEFM    'WISH IN THIS ROUTINE.'
7E7Ø ØØ        ØØ76Ø MSGND1    DEFB    ØH
7E71 4E        ØØ77Ø MESS2     DEFM    'NOW HIT ENTER TO RETURN TO BASIC'
7E91 ØØ        ØØ78Ø MSGND2    DEFB    ØH
7E92 E5        ØØ79Ø MANINT    PUSH    HL                 ;SAVE 'HL' REGISTERS.
7E93 2A157E    ØØ8ØØ           LD      HL,(CNTRL)         ;GET 2 PUSH INSTRUCTIONS.
7E96 E5        ØØ81Ø           PUSH    HL                 ;SAVE PUSH INSTRUCTIONS.
7E97 21C9ØØ    ØØ82Ø           LD      HL,ØC9H            ;ØC9H IS RETURN INSTR.
7E9A 22157E    ØØ83Ø           LD      (CNTRL),HL         ;PUT RETURN IN INT. PROC.
7E9D FB        ØØ84Ø           EI                         ;ENABLE INTERRUPTS.
7E9E F5        ØØ85Ø           PUSH    AF                 ;SAVE 'AF' REGISTERS.
7E9F C5        ØØ86Ø           PUSH    BC                 ;SAVE 'BC' REGISTERS.
7EAØ D5        ØØ87Ø           PUSH    DE                 ;SAVE 'DE' REGISTERS.
7EA1 21ØØ3C    ØØ88Ø           LD      HL,3CØØH           ;START OF SCREEN.
7EA4 11ØØ78    ØØ89Ø           LD      DE,78ØØH           ;START OF BUFFER.
7EA7 Ø1ØØØ4    ØØ9ØØ           LD      BC,4ØØH            ;# OF BYTES ON SCREEN.
7EAA EDBØ      ØØ91Ø           LDIR                       ;SAVE SCREEN DISPLAY.
7EAC CDE47E    ØØ92Ø           CALL    CLRSCN             ;CLEAR SCREEN.
7EAF 214Ø7E    ØØ93Ø           LD      HL,MESS1           ;START OF MESSAGE 1.
7EB2 Ø13ØØØ    ØØ94Ø           LD      BC,MSGND1-MESS1    ;LENGTH OF MESS1.
7EB5 11ØØ3C    ØØ95Ø           LD      DE,3CØØH           ;START OF SCREEN.
7EB8 EDBØ      ØØ96Ø           LDIR                       ;PRINT IT.
7EBA 21717E    ØØ97Ø           LD      HL,MESS2           ;START OF MESSAGE 2.
7EBD Ø12ØØØ    ØØ98Ø           LD      BC,MSGND2-MESS2    ;LENGTH OF MESS2.
7ECØ 114Ø3C    ØØ99Ø           LD      DE,3C4ØH           ;START OF NEXT LINE.
7EC3 EDBØ      Ø1ØØØ           LDIR                       ;PRINT IT.
7EC5 3A4Ø38    Ø1Ø1Ø KYSTRK    LD      A,(384ØH)          ;LOOK FOR ENTER KEY.
7EC8 FEØØ      Ø1Ø2Ø           CP      Ø                  ;TEST FOR ENTRY.
7ECA 2ØØ2      Ø1Ø3Ø           JR      NZ,RETURN          ;RETURN IF KEY ENTRY.
7ECC 18F7      Ø1Ø4Ø           JR      KYSTRK             ;LOOK AGAIN.
7ECE 21ØØ78    Ø1Ø5Ø RETURN    LD      HL,78ØØH           ;START OF BUFFER.
7ED1 11ØØ3C    Ø1Ø6Ø           LD      DE,3CØØH           ;START OF SCREEN DISPLAY.
7ED4 Ø1ØØØ4    Ø1Ø7Ø           LD      BC,4ØØH            ;# OF BYTES ON SCREEN.
7ED7 EDBØ      Ø1Ø8Ø           LDIR                       ;PRINT IT.
7ED9 D1        Ø1Ø9Ø           POP     DE                 ;RESTORE 'DE' REGISTERS.
7EDA C1        Ø11ØØ           POP     BC                 ;RESTORE 'BC' REGISTERS.
7EDB F1        Ø111Ø           POP     AF                 ;RESTORE 'AF' REGISTERS.
7EDC F3        Ø112Ø           DI                         ;DISABLE INTERRUPTS.
7EDD E1        Ø113Ø           POP     HL                 ;GET 2 PUSH INSTRUCTIONS.
7EDE 22157E    Ø114Ø           LD      (CNTRL),HL         ;RESTORE INSTR. TO IPROC.
7EE1 FB        Ø115Ø           EI                         ;ENABLE INTERRUPTS.
7EE2 E1        Ø116Ø           POP     HL                 ;RESTORE 'HL' REGISTERS.
7EE3 C9        Ø117Ø           RET                        ;RETURN FROM ISR.
7EE4 21ØØ3C    Ø118Ø CLRSCN    LD      HL,3CØØH           ;START OF SCREEN.
7EE7 Ø1ØØØ4    Ø119Ø           LD      BC,4ØØH            ;# OF BYTES TO CLEAR.
7EEA 362Ø      Ø12ØØ CLR       LD      (HL),2ØH           ;PUT IN SPACES.
7EEC 23        Ø121Ø           INC     HL                 ;GET NEXT SCREEN LOC.
7EED ØB        Ø122Ø           DEC     BC                 ;COUNT DOWN BYTES.
7EEE 78        Ø123Ø           LD      A,B                ;TEST IF THE
7EEF B1        Ø124Ø           OR      C                  ;BYTE COUNT IS ZERO.
7EFØ 2ØF8      Ø125Ø           JR      NZ,CLR             ;CONTINUE IF NOT.
7EF2 C9        Ø126Ø           RET                        ;RETURN WHEN DONE.
7DØØ           Ø127Ø           END     START
ØØØØØ TOTAL ERRORS
```

Reverse Video Hardware Modification

by Dan Placido

Everyone needs a change of pace, and reverse video offers you just that. Reverse video adds pizazz to action games such as Air Raid and Saucer and offers welcome relief from a screenful of dreary text. I viewed the acquisition of reverse video as the first step toward improved graphics capability.

A study of the TRS-80 technical manual proved what I had suspected: Getting reverse video is surprisingly easy. Dot data, which combines with composite synchronization pulses to form the composite video signal, is available at Z30 pin 1. For regular video, the dot-data signal is inverted by Z41, creating a normally-low output signal which goes high whenever a dot is present. If I could invert this signal before it reached Z41, the result would be a high signal which would go low whenever a dot was present—reverse video! If that inverter could be switched on and off at will, then I would have the best of both worlds.

The actual modification for reverse video consists of installing a switched inverter in the video signal path. Because the inverter used for the modification already exists inside the TRS-80, the process is relatively simple. Be advised before you consider reverse video that this is a hardware modification which will void your warranty.

Installing Reverse Video

I chose the header/connector cable assembly for this project, because it would facilitate the addition of other switches and still allow easy removal of

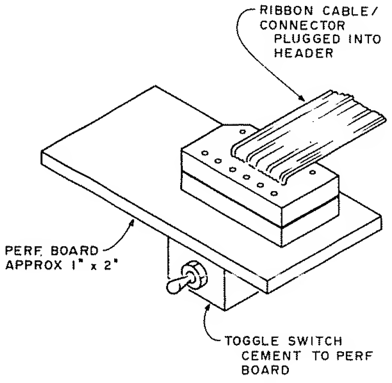

RIBBON CABLE/ CONNECTOR PLUGGED INTO HEADER
PERF. BOARD APPROX 1" x 2"
TOGGLE SWITCH CEMENT TO PERF BOARD

Figure 1. *Component layout and assembly*

the cover (by disconnecting the header). The cable is a 16-pin DIN cable (Radio Shack #276-1976) with one connector cut off. The header is a 16-pin IC socket (Radio Shack #276-1998). I cemented an SPST toggle switch and the header on a small section of perfboard, then wired pins 9 through 16 on the header together. After wiring the switch to pins 1, 2, and 3, I drilled a mounting hole in the cover and installed the assembly using the switch hardware. (See Figure 1.) The location I chose seemed to be a logical place because it offered easy access without being in the way.

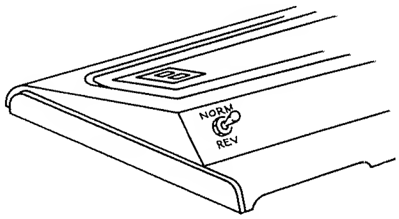


Figure 2. *Cover showing the switch location. The assembly is secured to the cover with the switch hardware.*

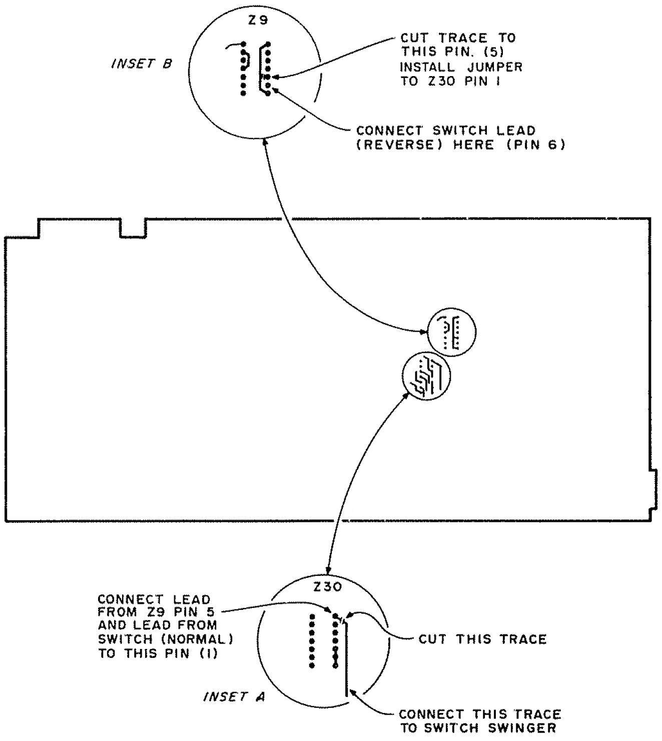

Z9
INSET B
CUT TRACE TO THIS PIN. (5) INSTALL JUMPER TO Z30 PIN 1
CONNECT SWITCH LEAD (REVERSE) HERE (PIN 6)
Z30
CONNECT LEAD FROM Z9 PIN 5 AND LEAD FROM SWITCH (NORMAL) TO THIS PIN (1)
CUT THIS TRACE
INSET A
CONNECT THIS TRACE TO SWITCH SWINGER

Figure 3. *Approximate location of Z9 and Z30 on the PC board (bottom view). Insets depict traces to be cut and connecting points.*

The inverter I used is a spare one located at Z9. I located Z30 on the main PC board and cut the trace at pin 1. (See Figure 3, inset A.) Likewise, I cut the trace at Z9 pin 5. (See Figure 3, inset B.) Taking the cable, I plugged one end into the header and cut off the other connector. I then separated the wires back about four or five inches and stripped and tinned all the leads. Following the cable color code (Figure 4), I soldered the wires from pins 9 through 16 to ground. (This serves as a shield from noise and crosstalk.) I soldered the wire from connector pin 1 to the trace of Z41 pins 6 and 7; the wire from pin 2 to Z30 pin 1; and the wire from pin 3 to pin 6 of Z9. Finally, I connected a jumper from Z30 pin 1 to Z9 pin 5. The wires from pins 4 through 8 are covered and tied back for future use. (See Figure 5.)

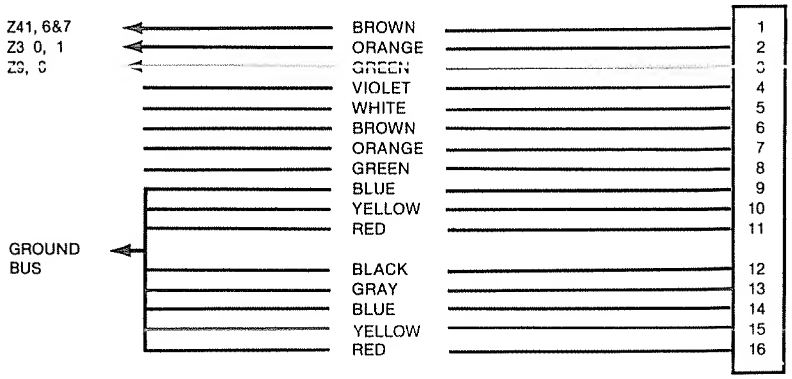


Figure 4. *Cable assembly. Notice that the colors repeat. Use care to select the correct wires. Pin 1 on the connector is identified by a beveled corner.*

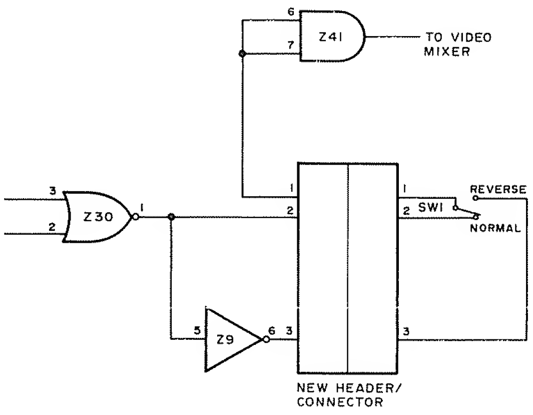


Figure 5. *Schematic*

Once you've achieved reverse video, some readjustment of the brightness and contrast will be necessary. On some sets, a slight vertical distortion may occur. This causes a slow, horizontal waving of the display. To remedy this replace R13 (a 2.2k Ohm resistor), located on the component board inside the video monitor, with a 49k Ohm resistor. The component board can be identified by the video-input cable connection. When attempting this or any other modification, remember that success depends upon good soldering practice and extreme care, especially when you are cutting traces.

HOME APPLICATIONS

Team Stats

by Robert J. Mott Jr.

I have developed a baseball/softball statistics program to use with a Model III 16K TRS-80 and a Radio Shack Line Printer II. As a result, I have established a statistical service for local teams. The idea is simple: I put each team on a cassette, then charge a modest fee for each update of their file. It's all done by mail. Each team gets two copies of the report (see Example 1), a blank worksheet (see Example 2), and a stamped, addressed envelope to mail the completed worksheet after their next game. I take the new information from the worksheet and update their file, producing two copies of the new report and a blank worksheet.

The report provides what I consider to be the most important offensive statistics for a team of up to 20 players. After the file is set up, eight entries are made on the 10-key pad of the Model III for each active player per update. Data entry takes no more than a few minutes per team.

The default setting is zero for all entries unless you are correcting an error, and the program gives you plenty of chances to do that. Most entries are self explanatory (1B = singles, 2B = doubles, and so on). But a few words about baseball statistics might be in order for those of you who are not familiar with them.

ABs, or at bats, means official at bats only. Hit batsmen, bases on balls, and sacrifices (all entered as a W) are not considered official at bats. The on-base (OB) percentage is determined by adding the Ws to the number of hits and dividing this number by the total of ABs and Ws. Slugging (SLG) percentage is the total number of bases the player reached on base hits (one for a single, two for a double, and so on) divided by the number of official at bats. There is a multiplier of 1000 (variable M) in lines 5420–5455 of the Program Listing to accommodate the CRT format. There are also checks which prevent I/O errors that could occur in file setup or if a player received a base on balls during the game but had no official at bats. By the way, K means strikeout in baseball lingo.

The program itself will allow you to do three things: You can set up files, update files, and print files. Setting up the files is the most difficult part. When you select the NEW FILE option, the system requests a team name. The team name is the file name. It must be exact, because when a file is read from tape for an update, the computer will accept only the precise name before reading the cassette. This prevents inadvertently loading the wrong file. Figure 1 shows the flowchart for the program.

```
                                                                BAT    SLG    OB     K
PLAYER          NO   AB    H   1B   2B   3B   HR   K    W   RBI   AVG    PCT    PCT    PCT
===============================================================================

BAKER            1   15    5    2    1    0    2    4    0    3   333    800    333    266

BIRD             2   13    3    2    0    1    0    2    3    2   230    384    375    153

COMISKEY         3    8    1    1    0    0    0    1    4    0   125    125    416    125

CROCKER          4   14    6    3    2    0    1    3    1    6   428    785    466    214

MCCARTHY C       5    4    0    0    0    0    0    3    0    0     0      0      0    750

MCCARTHY N       6   10    2    1    1    0    0    1    0    0   200    300    200    100

LORD             7    4    1    0    1    0    0    1    0    0   250    500    250    250

PUTNAM           8   12    5    1    2    2    0    1    2    2   416    916    500     83

===============================================================================

TEAM TOTALS          80   23   10    7    3    3   16   10   13   287    562    366    200
```

Example 1. *Sample report*

When you set up the file, use the proper player sequence. I use alphabetical order, but player number or any other sequence is acceptable. Once the file is established, you may add a player, but you may not change the sequence without going through a new file setup.

```
                                    WORKSHEET

PLAYER                             NO    AB   1B   2B   3B   HR   K    W    RBI
################################################################################
BAKER                              1    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
BIRD                               2    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
COMISKEY                           3    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
CROCKER                            4    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
MCCARTHY C                         5    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
MCCARTHY N                         6    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
LORD                               7    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
PUTNAM                             8    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
                                   0    ¶    ¶    ¶    ¶    ¶    ¶    ¶    ¶
================================================================================
```

Example 2. *Sample worksheet*

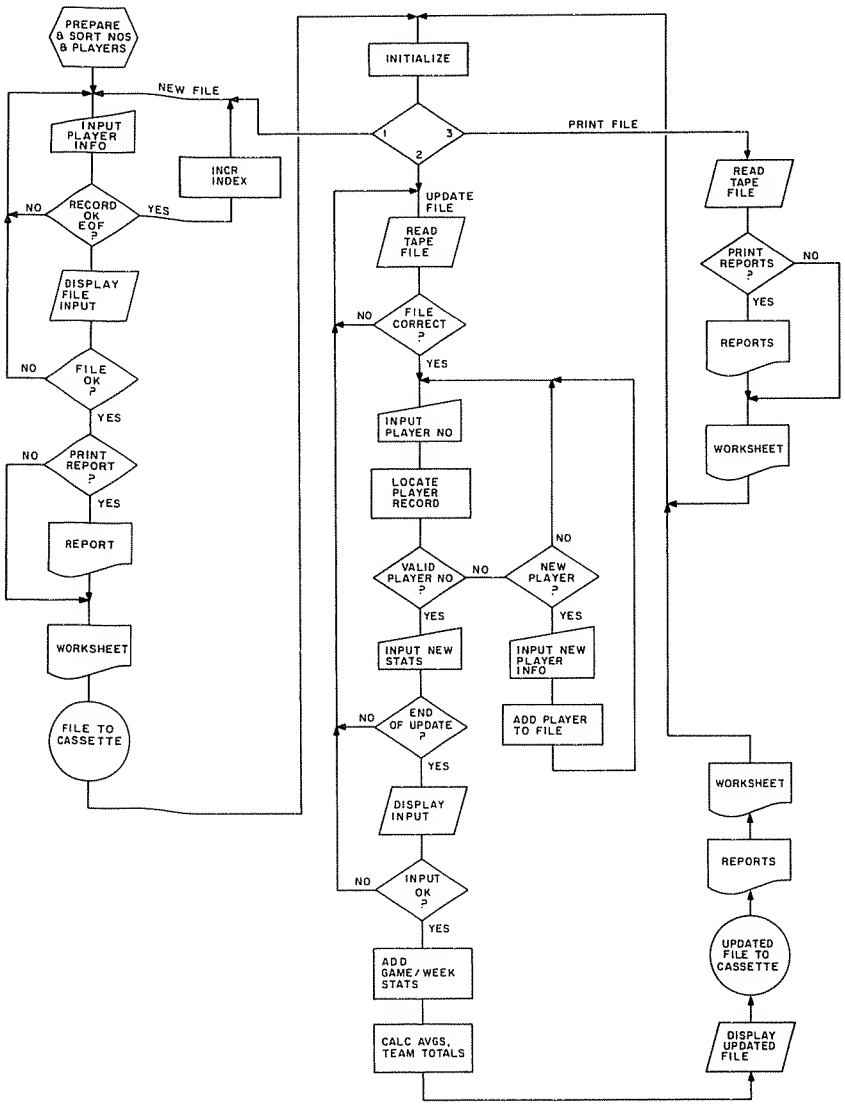


Figure 1. *Flowchart*

One 21-by-14 array is used as you build the file, although the zero subscript holds no player information. The first player's statistics are stored at subscript (1,N), the next at (2,N), etc. I refer to the first subscript as the index (I) throughout the program. There must not be any index without a player name until the end of the file. END OF FILE is for tape reading and writing keys from the null set in P(I). Each player must have a number, even if you assign them arbitrarily (up to 999), because during the updates, a search for the player number fetches the proper index for new player information. During file buildup or updates you can reset the index to change any player record until you enter Y after FILE OK--Y/N.

You may set up a team file with only player names and numbers to generate a team worksheet or wait for game statistics to set up the file. In the latter case you will have to tell team managers what information you need. Once you have the file set up, each data entry will produce a blank worksheet.

The PRINT FILE option merely reads the file from tape and asks how many copies of the report you want printed. Only one worksheet will be printed. I generally give two copies of the report to each manager. The FILE UPDATE option reads the old file from tape and stores it in array B%(21,14). A CONT UPDATE check makes sure you have the latest file. Once the proper file is read, you may enter the player number from the worksheet and enter the latest game or week figures. The program checks for valid player numbers and allows for the addition of new players at this point. Once all the new information is entered in array A%, all file input can be validated, then combined with previous data to produce an updated printout, worksheet, and tape.

The printed report uses the command TIME$. You must set TIME$ in DOS before entering or delete it from statement 7530 if DOS is not installed.

Program Listing. *Team Stats*

Encyclopedia Loader

```
  10 DEFINT S,P,I,Z,K,M,C,D,X,Y
  15 POKE 16425,1
  17 POKE 16424,65 :
     ' SET LINE TTL FOR PRINTER
  20 CLEAR 5000
  25 Q$ = "PCT":
     M = 1000
  30 DIM P$(21),A%(21,14),B%(21,14),T%(14)
  35 DP$ = CHR$(27) + CHR$(14) :
     ' DOUBLE SIZE CHAR
  40 POKE 16916,0
1000 CLS :
     S = 0:
     FOR N = 1 TO 5:
      PRINT :
      NEXT N:
     PRINT TAB(15) "T E A M   S T A T I S T I C S"
1001 PRINT :
     PRINT TAB(20) "NEW FILE = 1":
     PRINT TAB(20) "UPDATE FILE = 2":
     PRINT TAB(20) "PRINT FILE = 3"
1005 PRINT :
     PRINT TAB(30);:
     INPUT "SELECT: ";S
1010 ON S GOTO 2000,3000,4000
1020 GOTO 1000
2000 I = 1
2010 INPUT "NEW FILE--ENTER TEAM NAME";N$:
     IF N$ = "" GOTO 2010
2030 PRINT "I=";I:
     INPUT "RESET INDEX - IF END OF FILE, SET = 21, ELSE PRESS ENTER"
     ;I:
     IF I < 1 OR I > 21
      THEN
       GOTO 2030
2040 IF I = 21 GOTO 2200
2050 INPUT "PLAYER NAME";P$(I)
2070 INPUT "PLAYER NO";A%(I,1)
2090 GOSUB 6000
2100 X$ = "":
     INPUT "RECORD OK--Y/N";X$:
     IF X$ < > "Y" GOTO 2030
2105 IF NP$ = "Y" GOTO 3100
2110 I = I + 1:
     GOTO 2030
2200 :
     ' PRINT NEW DATA
2210 GOSUB 5000 :
     ' PRINT HEADING
2230 GOSUB 5400 :
     ' TOTAL & CALC PLAYER AVGS
2280 GOSUB 5100 :
     ' PRINT BODY A
2300 X$ = "N":
     INPUT "FILE OK--Y/N";X$
2310 IF X$ = "Y" GOSUB 5600:
     GOSUB 5200:
     GOSUB 7000:
     GOSUB 7600:
     GOTO 20 :
     ' CALC TEAM AVGS,LPRINT REPORT-WORKSHEET & UPDATE TP
2320 FOR S = 2 TO 10 :
      ' CLEAR TTLS
2330  T%(S) = 0
2340  NEXT S
2350 GOTO 2030 :
     ' CORRECT INPUT
3000 GOSUB 5300 :
```

Program continued

```
      ' READ FILE TO B%
3050 GOSUB 5000:
     GOSUB 5900 :
     ' PRINT HEADING-BODY B
3090 X$ = "N":
     INPUT "CONT. UPDATE--Y/N";X$:
     IF X$ < > "Y" GOTO 20
3092 FOR I = 1 TO 20 :
      ' MOVE PLAYER NO TO ARRAY A%
3094  A%(I,1) = B%(I,1)
3096  NEXT I
3100 I = 1:
     NP$ = "N" :
     ' SET INDEX & DEFAULT
3110 INPUT "ENTER PLAYER NO--999 TO END";A%(0,1)
3120 IF A%(0,1) = 999 GOSUB 5000:
     GOSUB 5100:
     GOTO 3500:
     '  CHECK FOR END OF UPDATE-PRINT INFO
3130 IF B%(I,1) = A%(0,1) GOTO 3210 :
     ' IF PLAYER NO FOUND GET INPUT
3150 I = I + 1
3170 IF P$(I) = "" PRINT "INVALID PLAYER NO":
     INPUT "NEW PLAYER--Y/N";NP$:
     IF NP$ < > "Y" GOTO 3100 :
      ELSE
       GOTO 2030 :
       ' CHECK FOR VALID PLAYER NO
3190 GOTO 3130 :
     ' LOOP THROUGH ALL PLAYERS
3210 PRINT P$(I):
     GOSUB 6000 :
     ' PRINT PLAYER NAME GET INPUT
3300 X$ = "N":
     INPUT "RECORD OK--Y/N";X$ :
     ' VERIFY INPUT
3305 IF X$ < > "Y" GOSUB 6000:
     GOTO 3300
3310 GOTO 3100 :
     ' NEXT PLAYER
3500 X$ = "N":
     INPUT "FILE OK--Y/N";X$:
     IF X$ < > "Y" GOTO 3100 :
     ' VERIFY FILE
3510 GOSUB 6200 :
     ' ADD NEW DATA
3515 GOSUB 5400 :
     ' TOTAL & CALC PLAYER AVGS
3520 GOSUB 5000:
     GOSUB 5100 :
     ' PRINT HEAD & BODY A
3525 GOSUB 5600 :
     ' CALC TEAM AVGS
3530 GOSUB 5200 :
     ' UPDATE FILE TO TP
3535 GOSUB 7000 :
     ' LPRINT REPORT
3550 GOSUB 7600 :
     ' PRINT WORKSHEET
3590 GOTO 20
4000 :
     ' READ FILE & LPRINT
4005 GOSUB 5300 :
     ' READ FILE
4010 GOSUB 5000 :
     ' PRINT HEADINGS
4020 GOSUB 6400 :
     ' TRANSFER B>A
4030 GOSUB 5100 :
     ' PRINT BODY A%
4040 GOSUB 5400 :
      ' TTL
4050 GOSUB 5600 :
      ' CALC TEAM AVGS
4080 GOSUB 7000 :
      ' LPRINT REPORT
4090 GOSUB 7600 :
      ' LPRINT WORKSHEET
4099 GOTO 20
5000 :
      ' PRINT HEADINGS
5001 CLS :
      POKE 16916,3 :
      ' SET SCROLL PROTECT
5010 Z = 32 - ( LEN(N$) / 2)
5030 PRINT TAB(Z)N$
5050 PRINT TAB(45)"BAT"; TAB(50)"SLG"; TAB(55)"OB"; TAB(60)"K"
5070 PRINT "PLAYER"; TAB(12)"NO"; TAB(16)"AB"; TAB(20)"H"; TAB(22)"1B
     "; TAB(25)"2B"; TAB(28)"3B"; TAB(31)"HR"; TAB(35)"K"; TAB(38)"W"
     ; TAB(40)"RBI"; TAB(45)"AVG"; TAB(50)Q$; TAB(55)Q$; TAB(60)Q$
5090 PRINT STRING$(64,"=");
5099 RETURN
5100 :
      ' PRINT BODY A%
5101 I = 1
5110 IF P$(I) = "" GOTO 5199
5130 PRINT USING "%          %";P$(I);
5131 PRINT USING "###";A%(I,1);
5132 PRINT USING "####";A%(I,2);
5133 PRINT USING "###";A%(I,3);A%(I,4);A%(I,5);A%(I,6);A%(I,7);A%(I,8
     );A%(I,9);
5134 PRINT USING "####";A%(I,10);
5135 PRINT USING "#####";A%(I,11);A%(I,12);A%(I,13);A%(I,14)
5150 I = I + 1
5170 IF I < 21 GOTO 5110
5199 RETURN
5200 :
      ' TAPE FILE UPDATE
5201 I = 1:
     INPUT "FILE UPDATE--PREPARE TAPE TO WRITE--PRESS ENTER";X
5203 PRINT # - 1,N$
5205 PRINT # - 1,P$(I),A%(I,1),A%(I,2),A%(I,3),A%(I,4),A%(I,5),A%(I,6
     ),A%(I,7),A%(I,8),A%(I,9),A%(I,10),A%(I,11),A%(I,12),A%(I,13),A%
     (I,14)
5230 IF P$(I) = "" GOTO 5299
5250 I = I + 1:
     GOTO 5205
5260 FOR I = 2 TO 14
5270  PRINT # - 1,T(I);
5280  NEXT I
5299 RETURN
5300 :
      ' READ FILE
5301 I = 1:
     INPUT "PREPARE TAPE TO READ--READ FILE FOR WHAT TEAM";T$
5310 INPUT # - 1,N$:
     IF T$ < > N$ GOTO 5301
5350 INPUT # - 1,P$(I),B%(I,1),B%(I,2),B%(I,3),B%(I,4),B%(I,5),B%(I,6
     ),B%(I,7),B%(I,8),B%(I,9),B%(I,10),B%(I,11),B%(I,12),B%(I,13),B%
     (I,14)
5360 IF P$(I) = "" GOTO 5399
5370 I = I + 1:
     IF I < 21 GOTO 5350
5399 RETURN
5400 I = 1 :
      ' TOTAL & CALC PLAYER AVGS
5405 IF A%(I,2) = 0 GOTO 5450
5410 A%(I,3) = A%(I,4) + A%(I,5) + A%(I,6) + A%(I,7) :
      ' HITS
5420 A%(I,11) = M * (A%(I,3) / A%(I,2)) :
      ' BAT AVG
5430 A%(I,12) = ((4 * A%(I,7) + 3 * A%(I,6) + 2 * A%(I,5) + A%(I,4))
```

Program continued

```
      / A%(I,2)) * M:
      ' SLG PCT
5440 A%(I,14) = A%(I,8) / A%(I,2) * M :
      ' K PCT
5450 IF A%(I,2) = 0 AND A%(I,9) = 0 GOTO 5490
5455 A%(I,13) = ((A%(I,3) + A%(I,9)) / (A%(I,2) + A%(I,9))) * M :
      ' OB PCT
5460 FOR K = 2 TO 10
5470  T%(K) = T%(K) + A%(I,K)
5480  NEXT K
5490 I = I + 1:
     IF P$(I) = "" GOTO 5499
5495 IF I < 21 GOTO 5405
5499 RETURN
5600 :
      ' CALC TEAM AVGS
5601 IF T%(2) = 0 GOTO 5799
5605 T%(11) = (T%(3) / T%(2)) * M
5610 T%(12) = ((T%(7) * 4 + T%(6) * 3 + T%(5) * 2 + T%(4)) / T%(2))
     * M
5630 T%(13) = ((T%(3) + T%(9)) / (T%(2) + T%(9))) * M
5650 T%(14) = (T%(8) / T%(2)) * M
5700 :
      ' PRINT TOTALS
5701 PRINT TAB(0) STRING$(64,"=");
5710 PRINT "TEAM TOTALS  ";
5711 PRINT USING "####";T%(2);T%(3);
5712 PRINT USING "###";T%(4);T%(5);T%(6);T%(7);T%(8);T%(9);
5713 PRINT USING "####";T%(10);
5714 PRINT USING "#####";T%(11);T%(12);T%(13);T%(14)
5799 RETURN
5900 :
      ' PRINT BODY--B
5901 I = 1
5910 IF P$(I) = "" GOTO 5999
5930 PRINT USING "%            %";P$(I);
5935 PRINT USING "###";B%(I,1);
5940 PRINT USING "####";B%(I,2);
5945 PRINT USING "###";B%(I,3);B%(I,4);B%(I,5);B%(I,6);B%(I,7);B%(I,8
     );B%(I,9);
5950 PRINT USING "####";B%(I,10);
5955 PRINT USING "#####";B%(I,11);B%(I,12);B%(I,13);B%(I,14)
5960 I = I + 1
5970 IF I < 21 GOTO 5910
5999 RETURN
6000 :
      ' INPUT NEW DATA
6001 INPUT "AB";A%(I,2)
6010 INPUT "1B";A%(I,4)
6030 INPUT "2B";A%(I,5)
6050 INPUT "3B";A%(I,6)
6070 INPUT "HR";A%(I,7)
6090 INPUT "K";A%(I,8)
6110 INPUT "W";A%(I,9)
6130 INPUT "RBI";A%(I,10)
6199 RETURN
6200 :
      ' ADD NEW DATA
6201 FOR I = 1 TO 20
6210  FOR K = 2 TO 10
6230   A%(I,K) = A%(I,K) + B%(I,K)
6250   NEXT K
6370  NEXT I
6399 RETURN
6400 :
      ' TRANSFER B>A
6401 FOR I = 1 TO 20
6410  FOR K = 1 TO 14
6420   A%(I,K) = B%(I,K)
6430   NEXT K
6440  NEXT I
```

```
6499 RETURN
7000 :
     ' LPRINT REPORT
7001 INPUT "PRINT HOW MANY COPIES";C
7005 IF C = 0 GOTO 7499
7010 FOR Y = 1 TO C
7020  FOR X = 1 TO 3
7030   LPRINT CHR$(138)
7040   NEXT X
7050  GOSUB 7500
7215  Z = 40 - LEN(N$)
7220  LPRINT TAB(Z); CHR$(27); CHR$(14);N$
7230  LPRINT CHR$(138); CHR$(138)
7250  LPRINT TAB(57)"BAT"; TAB(63)"SLG"; TAB(69)"OB"; TAB(75)"K"
7260  LPRINT "PLAYER"; TAB(15)"NO"; TAB(20)"AB"; TAB(25)"H";
      TAB(28)"1B"; TAB(32)"2B"; TAB(36)"3B"; TAB(40)"HR"; TAB(44)"K";
       TAB(48)"W"; TAB(51)"RBI"; TAB(57)"AVG"; TAB(63)Q$; TAB(69)Q$;
      TAB(75)Q$
7290  LPRINT STRING$(80,"=")
7300  LPRINT CHR$(138):
      I = 1
7305  IF P$(I) = "" GOTO 7400
7310  LPRINT USING "%              %";P$(I);
7320  LPRINT USING "####";A%(I,1);
7330  LPRINT USING "#####";A%(I,2);
7340  LPRINT USING "####";A%(I,3);A%(I,4);A%(I,5);A%(I,6);A%(I,7);A%(
      I,8);A%(I,9);A%(I,10);
7360  LPRINT USING "######";A%(I,11),A%(I,12),A%(I,13),A%(I,14)
7365  LPRINT CHR$(138)
7370  I = I + 1:
      IF I < 21 GOTO 7305
7400  LPRINT STRING$(80,"=")
7410  LPRINT CHR$(138);"TEAM TOTALS          ";
7420  LPRINT USING "####";T%(2);T%(3);T%(4);T%(5);T%(6);T%(7);T%(8);T
      %(9);T%(10);
7430  LPRINT USING "######";T%(11);T%(12);T%(13);T%(14)
7440  LPRINT CHR$(12)
7450  NEXT Y
7499 RETURN
7500 :
     ' LPRINT DDP HEAD
7501 LPRINT TAB(18) DP$;"HORNSBY'S HOME RUN"
7510 LPRINT TAB(18)"YANKEE STADIUM"; TAB(43)"NEW YORK, NY 10017"
7530 LPRINT TAB(18)"555-1212"; TAB(45) TIME$
7550 LPRINT CHR$(138); CHR$(138)
7599 RETURN
7600 :
     ' PRINT WORKSHEET
7601 I = 1:
     GOSUB 7500
7610 LPRINT TAB(30)DP$;"WORKSHEET"
7630 LPRINT CHR$(138); CHR$(138)
7650 LPRINT "PLAYER"; TAB(32)"NO"; TAB(39)"AB"; TAB(44)"1B";
     TAB(49)"2B"; TAB(54)"3B"; TAB(59)"HR"; TAB(64)"K"; TAB(69)"W";
     TAB(74)"RBI"
7670 LPRINT STRING$(80,"#")
7680 LPRINT P$(I); TAB(31)A%(I,1);
7690 LPRINT TAB(37) CHR$(124); TAB(42) CHR$(124); TAB(47) CHR$(124);
     TAB(52) CHR$(124); TAB(57) CHR$(124); TAB(62) CHR$(124);
     TAB(67) CHR$(124); TAB(72) CHR$(124)
7730 LPRINT STRING$(80,"=")
7740 IF P$(I) = "" GOTO 7790
7750 I = I + 1:
     IF I < 21 GOTO 7680
7790 LPRINT CHR$(12)
7799 RETURN
```

Loans—Do You Really Know the Cost of Yours?

by Tom Van Dan Elzen

Banks and other lending institutions are run by humans, and humans make mistakes. An error of only $1.00 per payment on a 30-year loan amounts to $360.00. I've checked loans for friends and have found that about eight out of ten have errors—as small as $.18 per payment, and as high as $2.90 per payment. (The $2.90 error was on a 25-year loan for a total of $870.00.)

In a local Sunday paper, an advertisement for a new car from a local car dealer went something like this: "Anniversary Sale. Buy a new Bananaboat for only $158.00 per month." At the bottom of the ad were the particulars: 48 months at $158.00 per month. APR (Annual Percentage Rate) 14.98 percent. Amount financed $5,632.16. Finance charges $1,951.84. Selling price $6,136.16. Total amount of payments $7,584.00. If you run these figures through the Loans program (see Program Listing) and solve for payment, the monthly payment is $156.69 ($1.31 less than advertised). Over 48 months, this amounts to an overpayment of $62.88. If you assume that the stated payment is correct and solve for the APR, the result is 15.438 percent (almost 1/2 percent higher than stated). One possible reason for the discrepancy is the length of time between the signing of the contract and the first payment. If it is 45 days instead of the standard one month, this could account for the increase. Remember this when you see an advertisement like this:

SALE—SOLID PLASTIC FURNITURE
No payments for 90 Days
Happy Joe's Furniture

Happy Joe is so happy because, during those 90 days, the interest accrues, and the cost of the loan grows rapidly. The Loans program assumes that the first payment is due one month after the contract is consummated.

There are five basic parts to any loan:

- The payment—The amount you contract to pay to the loan company each month.
- APR—The Annual Percentage Rate. This is the interest rate specified by the loan company.
- Principal—The original amount of the loan.
- The number of payments—A four-year loan = 48 payments; a 25-year loan = 300 payments.

● Balloon—A balloon payment is usually associated with short-term contracts under which a given amount of principal is borrowed for a given number of months at a specified APR, but at relatively small monthly payments (payments too small to pay off the loan). At the end of the agreed number of payments, the remaining balance is due in total. This balance is called the balloon payment. You can use this routine to calculate the early payoff balance on your loans. For example, if you want to pay off your four-year car loan at the end of the third year, solve for the balloon payment.

All through the program, you will see the previous entries or calculations for the five parts of the loan displayed in brackets just before the ?. If the previous APR you entered was 12 percent, the display would be:

```
ENTER A.P.R. <12>?
```

This serves as a scratchpad so you can follow changes without pencil and paper.

When you calculate any of the five parts of a loan, the screen displays Total Cost of the Loan and the Total Interest Paid over the life of the loan. This helps you to determine the best arrangements you can make for a new loan.

There is a message at the bottom of each screen which tells you that pressing M returns you to the menu and that pressing the up arrow does the present routine over.

How the Program Works

The program is made up of a menu and seven subroutines. The menu (see Figure 1) gives you a choice of seeing a set of definitions of terms, solving for any one of the five parts of a loan, or using an extra routine to calculate the total interest paid for each year. The subroutines are as follows.

1) Definition of Terms (see Figures 2, 3, and 4)—Selecting Definition of Terms from the menu gives you a brief description of each of the important inputs the program uses.

2) Monthly Payment (see Figure 5)—Selecting Monthly Payment from the menu sends the program to the second subroutine. It asks you to enter, one

```
                    *****  MENU  *****

1)  DEFINITION OF TERMS
2)  MONTHLY PAYMENT
3)  ANNUAL PERCENTAGE RATE (A.P.R.)
4)  NUMBER OF PAYMENTS
5)  PRINCIPAL BALANCE AFTER (XX) PAYMENTS. COMMONLY CALLED 'LOAN
    PAYOFF' OR 'BALLOON' PAYMENT.
6)  ORIGINAL PRINCIPAL. 'HOW MUCH CAN YOU AFFORD'?
7)  END OF YEAR INTEREST PAID. (FOR INCOME TAX PURPOSES)
ENTER YOUR CHOICE FROM THE MENU BY THE NUMBER?_
```

Figure 1

at a time, the amount of principal, the number of payments, the APR, and the balloon payment (if any).

```
*****DEFINITION OF TERMS*****
'PAYMENT' IS THE MONTHLY PAYMENT.
'PRINCIPAL' IS THE TOTAL AMOUNT OF THE LOAN INCLUDING ANY BALLOON.
'NUMBER OF PAYMENTS' IS THE TOTAL NUMBER OF MONTHS THAT THE LOAN
REQUIRES FOR REPAYMENT.
'A.P.R.' IS THE ANNUAL PERCENTAGE RATE. WHEN ENTERING THIS VALUE, DO
NOT ENTER IT AS A PERCENT. FOR EXAMPLE, 12.5% WOULD BE ENTERED AS 12.5,
NOT .125 OR 12.5%
PRESS ANY KEY TO CONTINUE
```

Figure 2

```
*****DEFINITION OF TERMS CONTINUED*****
'BALLOON'—THIS TERM IS BEST DEFINED BY EXAMPLE. SAY YOU WOULD LIKE
TO BORROW $5000.00 FOR 48 MONTHS AT 12% A.P.R., THIS WOULD REQUIRE A
MONTHLY PAYMENT OF $131.67.
HOWEVER, DUE TO OTHER SHORT TERM BILLS, YOU WOULD LIKE TO MAKE
SMALLER PAYMENTS (FOR EXAMPLE $100.00 PER MONTH). THIS MEANS THAT
WHEN PAYMENT 48 COMES DUE, THERE WILL STILL BE $1938.92 REMAINING.
THUS, THIS PAYMENT EXPANDS (LIKE A BALLOON) FROM $100.00 TO $1938.92.
PRESS ANY KEY TO CONTINUE
```

Figure 3

```
*****DEFINITION OF TERMS CONTINUED*****
  THE ROUTINE FOR CALCULATING A BALLOON PAYMENT MAY ALSO BE USED
TO CALCULATE AN EARLY PAYOFF ON A LOAN. THIS IS ACCOMPLISHED BY
ENTERING 'THE NUMBER OF PAYMENTS ALREADY MADE + 1' WHEN ASKED
'ENTER NUMBER OF PAYMENTS'?
THE RESULTING 'BALLOON' WILL BE THE 'PAYOFF' DUE ON THE NEXT
SCHEDULED PAYMENT.
PRESS 'M' FOR MENU
```

Figure 4

```
THIS PROGRAM COMPUTES MONTHLY PAYMENT

ENTER AMOUNT OF PRINCIPLE < 0 >? 100
ENTER NUMBER OF PAYMENTS < 0 >? 122
ENTER A.P.R. < 0 >? 12
ENTER BALLOON IF ANY < 0 >? 0

THE MONTHLY PAYMENT IS   $   1.42

THE TOTAL COST OF THE LOAN IS   $   173.24
THE TOTAL INTEREST PAID IS   $   73.24


PRESS 'M' FOR MENU OR ↑ TO RE-CALCULATE PAYMENT
```

Figure 5

3) APR (see Figure 6)—The program asks you to enter the four parts of a loan, other than the APR. At the top of the CRT, the number of loops that the routine is taking to calculate the APR appears. Unlike the other four parts of a loan, the APR does not have a formula for calculation. I use a binary approach to successive approximation. This takes a while; so the screen displays the number of iterations (loops) to let you know that it is still working. I've waited through a maximum of 25 loops. If you find yourself waiting longer, check your entries for an error.
4) Number of Payments (see Figure 7)—Again the program asks for the four missing elements of the loan. It computes the number of payments to one decimal place. If the answer is not a whole number (for example, 12.0), your last payment will be a partial payment.

```
THIS WILL CALCULATE THE A.P.R.

ENTER MONTHLY PAYMENT < 0 > ? 88.85
ENTER NUMBER OF PAYMENTS < 0 > ? 12
ENTER AMOUNT OF PRINCIPAL < 0 > ? 1000
ENTER BALLOON, IF ANY < 0 > ? 0

THE A.P.R. = 12.000%

THE TOTAL COST OF THE LOAN IS  $   1066.20
THE TOTAL INTEREST PAID IS  $   66.20


PRESS 'M' FOR MENU OR ↑ TO CALCULATE NEW A.P.R.
```

Figure 6

```
THIS WILL CALCULATE THE NUMBER OF PAYMENTS

ENTER MONTHLY PAYMENT < 0 > ? 88.85
ENTER A.P.R. < 0 > ? 12
ENTER AMOUNT OF PRINCIPAL < 0 > ? 1000
ENTER BALLOON, IF ANY < 0 > ? 0

THE NUMBER OF PAYMENTS ARE   12.0

THE TOTAL COST OF THE LOAN IS  $   1066.20
THE TOTAL INTEREST PAID IS  $   66.20

PRESS 'M' FOR MENU OR ↑ TO CALCULATE NEW NUMBER OF PAYMENTS
```

Figure 7

5) Balloon (see Figure 8)—You are again asked to enter the four missing elements for the loan. If you wish to calculate an early payoff on a loan, answer the Enter Number of Payments question with the number of payments you have made, not the number of payments the loan was contracted for.

6) Principal (see Figure 9)—Again you must enter the four missing parts; however, I have changed the wording to help you. I am assuming that the people who use this routine want to know how large a loan they can afford. I have worded the input questions accordingly.

```
THIS WILL CALCULATE THE 'LOAN PAYOFF' OR 'BALLOON' PAYMENT

ENTER MONTHLY PAYMENT < 88.85 > ? 88.85
ENTER A.P.R. < 12 > ? 12
ENTER AMOUNT OF PRINCIPAL < 1000 > ? 1000
ENTER NUMBER OF PAYMENTS < 12 > ? 12

THE 'BALLOON' OR 'PAYOFF' PAYMENT IS  $  0.00

THE TOTAL COST OF THE LOAN IS  $  1066.20
THE TOTAL INTEREST PAID IS  $  66.20

PRESS 'M' FOR MENU OR ↑ TO CALCULATE NEW BALLOON OR PAYOFF
```

Figure 8

```
CALCULATES HOW MUCH PRINCIPAL YOU CAN AFFORD

ENTER MONTHLY PAYMENT YOU CAN AFFORD < 88.85 >? 88.85
ENTER BEST A.P.R. AVAILABLE < 12 >? 12
ENTER NUMBER OF PAYMENTS YOU ARE WILLING TO MAKE < 12 >? 12
ENTER BALLOON, IF ANY < 0 >? 0

THE AMOUNT OF PRINCIPAL YOU CAN AFFORD IS  $  1000.00

THE TOTAL COST OF THE LOAN IS  $  1066.20
THE TOTAL INTEREST PAID IS  $  66.20

PRESS 'M' FOR MENU, ↑ TO CALCULATE A NEW PRINCIPAL
```

Figure 9

```
CALCULATES END OF YEAR INTEREST FOR INCOME TAX PURPOSES

ENTER MONTHLY PAYMENT < 0 >? 88.85
ENTER A.P.R. < 0 >? 12
ENTER AMOUNT OF PRINCIPAL < 0 >? 1000
ENTER NUMBER OF PAYMENTS (WHOLE NUMBERS ONLY) < 0 >? 12
ENTER BALLOON, IF ANY < 0 >? 0
ENTER NUMBER OF PAYMENTS IN THE FIRST YEAR < 0 >? 6_
```

Figure 10

7) End of Year Interest (see Figures 10, 11, and 12)—This routine asks for the five parts of the loan, plus the number of payments in the first year. Two points to watch are the number of payments (they must be in whole numbers) and the number of payments in the first year. If the first payment is due in August, for example, then the number of payments in the first year

is five (August, September, October, November, and December). After you enter the number of payments in the first year, the screen displays:
1) The total interest paid in the first year.
2) The total principal paid in the first year.
3) The remaining balance.

```
THE INTEREST PAID AT THE END OF YEAR < 1 > IS $   48.01
THE PRINCIPAL PAID AT THE END OF YEAR < 1 > IS $   485.09
THE PRINCIPAL REMAINING AT THE END OF YEAR < 1 > IS $   514.91

TO CONTINUE PRESS ANY KEY

TO ESCAPE THIS ROUTINE PRESS 'E'
```

Figure 11

```
THE INTEREST PAID AT THE END OF YEAR < 2 > IS $   18.17
THE PRINCIPAL PAID AT THE END OF YEAR < 2 > IS $   514.93
THE PRINCIPAL REMAINING AT THE END OF YEAR < 2 > IS $   -0.02

THE BANK OWES YOU $   0.02

PRESS 'M' FOR MENU, ↑ TO CALCULATE NEW YEAR END INTEREST
```

Figure 12

To continue, press any key. If you don't want to see the results for successive years, press E (for escape) and you will see the message at the bottom of the screen to press M (for menu) or the up arrow to calculate a new end of year interest. Pressing any other than those mentioned above will generate the results for the end of the second year, then the third year, and so on, until the end of the payments.

The Program

Lines 500 to 720 constitute the Menu routine. The variable, Menu, can be any number from 1 to 7. Line 720 sends you to subroutines 1000, 2000, 3000, 4000, 5000, 6000, or 7000 respectively.

Lines 1000 to 1460 make up the Definition of Terms routine. This is mostly text and PRINT commands.

Lines 2000 to 2400 contain the Monthly Payment subroutine. Other than a few error traps (such as lines 1100 and 1140), the payment is calculated by the textbook formula:

```
PMT = (PCT * (AMT - BAL * (1 + PCT) ↑ -NP)) / (1 - (1 + PCT) ↑ -NP)
```

PMT	=	Monthly payment
PCT	=	Monthly Interest Rate (APR/12)
AMT	=	Original Amount of Loan (Principal)
BAL	=	Balloon
NP	=	Number of Payments

Lines 3000 to 3520 are the APR calculations. APR cannot be found by a simple formula; so I have built a routine to calculate the APR using the binary approach to successive approximation. It works as follows:

1) First, I multiply the payment entered by the user by 100 to move the cents to the left of the decimal point; then I take the integer of this number to drop off any double precision residue from previous calculations. This whole number is saved as the variable PMT.

2) Next, I solve for a new payment (PMT (1)), using the other parts of the loan based on an APR of 100 percent. (It can't get that high, can it?)

3) The new payment is then converted to an integer in the same way as stated in the first step.

4) The two payments are then compared to each other.

5) If the new payment is greater than the true payment, the new APR is equal to: (the last low APR + the present APR) / 2.

6) If the new payment is less than the true payment, the new APR is equal to: (the last high APR + the present APR) / 2.

7) If the new payment is equal to the true payment, then the present APR must be the true APR.

8) If either step 5 or step 6 is true, the program jumps back to step 2 and continues until step 7 is true. When step 7 is true, the routine drops to line 3340 and prints out the APR calculated to three decimal places.

9) Lines 3500 and 3520 are an error trap to prevent division by 0 (/0?). This can occur if you enter erroneous data that leads to an APR of zero.

Lines 4000 to 4360 constitute the Number of Payments calculation subroutine. This follows the textbook formula for finding number of payments:

NP = LOG (PMT − (PCT * BAL)) / (PMT − (PCT * AMT)) / LOG (1 + PCT)

NP	=	Number of payments
PMT	=	Monthly payment
PCT	=	Monthly Interest (APR/12)
BAL	=	Balloon
AMT	=	Amount of Loan (Principal)

Lines 5000 to 5440 are the balloon calculation subroutine. This follows the textbook formula:

BAL = (AMT − (PMT * ((1 − (1 + PCT) ↑ −NP)) / PCT)) / (1 + PCT) ↑ −NP

Lines 6000 to 6380 are the subroutine that calculates the principal. This follows the formula:

AMT = (PMT * ((1 + PCT) ↑ −NP) /PCT)) + (BAL * (1 + PCT) ↑ −NP)

Lines 7000 to 7800 calculate the year-end interest and principal paid. This is based on the formulas:

Monthly Interest Paid = Monthly Interest Rate * New Balance
New Balance = Old Balance − (Payment − Interest Paid)

These two formulas are computed over and over so that:

Yearly Interest Paid = Sum of all the monthly interests paid in that year.
Yearly Principal Paid = Sum of all the monthly principals paid in that year.

Throughout the program, I have inserted spaces quite liberally for readability of the listing. All spaces (except those inside quotation marks) can be eliminated.

I did not write the subroutines with the intent of conserving memory, as 16K leaves plenty to work with. I wrote them to allow their use apart from the program. If you want to calculate only payments, for example, all you need are lines 2000 to 2300.

If you have a printer and would like a printout of the complete amortization schedule, make the following changes and additions.

```
7270 LPRINT "PAYMENT #","    INTEREST","    PRINCIPAL","    BALANCE":LPRINT " "

7390 A$ = "#######.##"
7392 LPRINT X + 1;
7394 PRINTTAB(12);:LPRINT ,;:LPRINTUSING A$;MI;
7396 PRINTTAB(29);:LPRINT,;:LPRINTUSING A$;MP;
7398 PRINTTAB(43);:LPRINT ,;:LPRINTUSING A$;AMT(1)
```

Note the number of spaces in line 7270. In addition, change all PRINT and PRINTUSING commands to LPRINT and LPRINTUSING in lines 7520, 7540, 7560, and 7580. In line 7700, change GOTO 7280 to GOTO 7270. See Figure 13 for a sample printout.

```
PAYMENT #   INTEREST   PRINCIPAL   BALANCE
1           0.83       9.17        90.83
2           0.76       9.24        81.59
3           0.68       9.32        72.27
4           0.60       9.40        62.87
5           0.52       9.48        53.40

THE INTEREST PAID AT THE END OF YEAR < 1 > IS $    3.40
THE PRINCIPAL PAID AT THE END OF YEAR < 1 > IS $   46.60
THE PRINCIPAL REMAINING AT THE END OF YEAR < 1 > IS $    53.40
```

Figure 13

The following five examples allow you to check your program for errors:

Example	Amount	APR	# of Payments	Payment	Balloon
1	$ 1,000.00	12%	12 months	$ 88.85	0
2	$10,000.00	13%	48 months	$268.28	−.10
3	$25,000.00	14%	15 years	$327.98	$3,002.67
4	$35,000.00	15%	25 years	$448.29	$ 3.12
5	$50,000.00	16%	30 years	$671.81	$4,974.94

You can use the above setup and substitute any four variables to solve for the fifth variable.

Example

AMT	Amount of principal
AMT(1)	Holding variable for new amount of principal calculations when I don't want the original amount (AMT) disturbed.
APR	Annual percentage rate
BAL	Balloon or balance
FY	Number of payments in first year
MENU	MENU selection
MI	Monthly interest paid
MP	Monthly principal paid
NP	Number of payments
PCT	Monthly interest in percentage
PCT(1)	Last high PCT (used in APR calculation)
PCT(2)	Last low PCT (used in APR calculation)
PMT	Monthly payment
PMT(1)	Calculated monthly payment used to compare to PMT in APR calculation.
YI	Yearly interest paid
YP	Yearly principal paid
X,Y,Z	Transient variables
X$	Used in all INKEY$ routines

Table 1. *Variable list*

Program Listing

Encyclopedia Loader

```
100 :
    '                    **********************
120 :
    '                    *                    *
140 :
    '                    *     WRITTEN BY     *
160 :
    '                    * T.J. VAN DAN ELZEN *
180 :
    '                    *       7/4/81       *
200 :
    '                    *                    *
220 :
    '                    **********************
240 :
    '
260 :
    '
500 :
    ' ***** MENU *****
520 CLS :
    PRINT "               *****  MENU  *****"
540 PRINT :
    PRINT :
    PRINT "1) DEFINITION OF TERMS"
560 PRINT "2) MONTHLY PAYMENT"
580 PRINT "3) ANNUAL PERCENTAGE RATE  (A.P.R.)"
600 PRINT "4) NUMBER OF PAYMENTS"
620 PRINT "5) PRINCIPAL BALANCE AFTER (XX) PAYMENTS. COMMONLY CALLED
         'LOAN PAYOFF' OR 'BALLOON' PAYMENT."
640 PRINT "6) ORIGINAL PRINCIPAL. 'HOW MUCH CAN YOU AFFORD' ?"
660 PRINT "7) END OF YEAR INTEREST PAID. (FOR INCOME TAX PURPOSES)"
680 PRINT @960,""; :
    INPUT "ENTER YOUR CHOICE FROM THE MENU BY THE NUMBER"; MENU
700 IF MENU < 1 OR MENU > 7 GOTO 520
720 ON MENU GOTO 1000,2000,3000,4000,5000,6000,7000
740 :
    '
760 :
    '
1000 :
    ' *****  DEFINITION OF TERMS  *****
1020 CLS :
    PRINT "          ***** DEFINITION OF TERMS *****
1040 PRINT :
    PRINT " 'PAYMENT' IS THE MONTHLY PAYMENT."
1060 PRINT :
    PRINT " 'PRINCIPAL' IS THE TOTAL AMOUNT OF THE LOAN INCLUDING AN

    Y        BALLOON."
1080 PRINT :
    PRINT " 'NUMBER OF PAYMENTS' IS THE TOTAL NUMBER OF MONTHS THAT
    THE      LOAN REQUIRES FOR REPAYMENT."
1100 PRINT :
    PRINT " 'A.P.R.' IS THE ANNUAL PERCENTAGE RATE. WHEN ENTERING TH
    IS       VALUE, DO NOT ENTER IT AS A PERCENT. FOR EXAMPLE, 12.5
    % WOULD  BE ENTERED AS 12.5, NOT .125 OR 12.5 %"
1120 PRINT @960, "PRESS ANY KEY TO CONTINUE";
1140 IF INKEY$ = ""
     THEN
      1140 :
     ELSE
     CLS
1160 PRINT "         ***** DEFINITION OF TERMS CONTINUED *****"
1180 PRINT :
    PRINT " 'BALLOON' - THIS TERM IS BEST DEFINED BY EXAMPLE. SAY YO
    U WOULD  LIKE TO BORROW $5000.00 FOR 48 MONTHS AT 12 % A.P.R., T
    HIS      WOULD REQUIRE A MONTHLY PAYMENT OF $131.67."
```

Program continued

```
1200 PRINT :
     PRINT "  HOWEVER, DUE TO OTHER SHORT TERM BILLS, YOU WOULD LIKE
     TO MAKE  SMALLER PAYMENTS (FOR EXAMPLE $100.00 PER MONTH).";
1220 PRINT " THIS MEANS    THAT WHEN PAYMENT 48 COMES DUE, THERE WILL
      STILL BE $1938.92    REMAINING. THUS, THIS PAYMENT EXPANDS (LIK
     E A BALLOON) FROM     $100.00 TO $1938.92."
1240 PRINT @960, "PRESS ANY KEY TO CONTINUE";
1260 IF INKEY$ = ""
      THEN
       1260 :
      ELSE
       CLS
1280 PRINT "         ***** DEFINITION OF TERMS CONTINUED *****
1300 PRINT :
     PRINT "  THE ROUTINE FOR CALCULATING A BALLOON PAYMENT MAY ALSO
     BE USED TO CALCULATE AN EARLY PAYOFF ON A LOAN.";
1320 PRINT " THIS IS ACCOMPLISHED BY ENTERING 'THE NUMBER OF PAYMENTS
      ALREADY MADE + 1' WHEN ASKED "
1340 PRINT :
     PRINT " 'ENTER NUMBER OF PAYMENTS' ?" :
     PRINT
1360 PRINT " THE RESULTING 'BALLOON' WILL BE THE 'PAYOFF' DUE ON THE
     NEXT    SCHEDULED PAYMENT."
1400 PRINT @960, "PRESS 'M' FOR MENU";
1420 X$ = INKEY$ :
     IF X$ = ""
      THEN
       1420
1440 IF X$ = "M"
      THEN
       500
1460 GOTO 1400
1480 :
     '
1500 :
     '
2000 :
     ' ***** CALCULATES MONTHLY PAYMENT *****
2020 CLS
2040 PRINT "THIS PROGRAM COMPUTES MONTHLY PAYMENT":
     PRINT :
     PRINT
2060 PRINT "ENTER AMOUNT OF PRINCIPLE <"AMT">"; :
     INPUT AMT
2080 PRINT "ENTER NUMBER OF PAYMENTS <"NP">"; :
     INPUT NP
2100 IF NP < 1 GOTO 2080
2120 PRINT "ENTER A.P.R. <"APR">"; :
     INPUT APR
2140 IF APR = 0 GOTO 2120
2160 PRINT "ENTER BALLOON IF ANY <"BAL">"; :
     INPUT BAL
2180 PCT = APR / 1200
2200 PMT = (PCT * (AMT - (BAL * ((1 + PCT) ↑ - NP)))) / (1 - ((1
     + PCT) ↑ - NP))
2220 IF (PMT * 100) - INT(PMT * 100) > .49
      THEN
       PMT = INT(PMT * 100) + 1 :
      ELSE
       PMT = INT(PMT * 100)
2240 PMT = PMT / 100
2260 PRINT :
     PRINT USING "THE MONTHLY PAYMENT IS   $#####.##"; PMT
2280 PRINT :
     PRINT USING "THE TOTAL COST OF THE LOAN IS  $#######.##"; (NP
     * PMT) + BAL
2300 PRINT USING "THE TOTAL INTEREST PAID IS  $#######.##";(NP
     * PMT) + (BAL - AMT)
2320 PRINT @ 960,"PRESS 'M' FOR MENU OR '↑' TO RE-CALCULATE PAYMENT";
2340 X$ = INKEY$ :
     IF X$ = ""
```

```
         THEN
          2340
2360 IF X$ = "M"
         THEN
          500
2380 IF X$ = "↑"
         THEN
          2020
2400 GOTO 2340
2420 :
     '
2440 :
     '
3000 :
     ' ***** CALCULATES A.P.R. (INTEREST) *****
3020 CLS :
     PCT = .0825 :
     PCT(2) = 0 :
     X = 0
3040 PRINT "THIS WILL CALCULATE THE A.P.R. ":
     PRINT
3060 PRINT "ENTER MONTHLY PAYMENT < "PMT"> "; :
     INPUT PMT
3080 PRINT "ENTER NUMBER OF PAYMENTS  <"NP"> "; :
     INPUT NP
3100 IF NP < 1 GOTO 3080
3120 PRINT "ENTER AMOUNT OF PRINCIPAL  <"AMT"> "; :
     INPUT AMT
3140 PRINT "ENTER BALLOON, IF ANY  <"BAL"> "; :
     INPUT BAL
3160 ON ERROR GOTO 3520
3180 PMT(1) = (PCT * (AMT - (BAL * ((1 + PCT) ↑ - NP)))) / (1
     - ((1 + PCT) ↑ - NP))
3200 PMT(1) = INT(PMT(1) * 1000) / 1000
3220 IF PMT(1) = PMT
         THEN
          3300
3240 IF PMT(1) > PMT
         THEN
          PCT(1) = PCT :
          PCT = (PCT + PCT(2)) / 2
3260 IF PMT(1) < PMT
         THEN
          PCT(2) = PCT :
          PCT = (PCT + PCT(1)) / 2
3280 X = X + 1 :
     PRINT @50,"LOOP";X; :
     GOTO 3180
3300 PRINT @ 512,"";
3320 APR = INT(PCT * 1200000) / 1000
3340 PRINT USING "THE A.P.R. = ##.###";APR;:
     PRINT " %"
3360 PRINT :
     PRINT USING "THE TOTAL COST OF THE LOAN IS  $#######.##"; (NP
     * PMT) + BAL
3380 PRINT USING "THE TOTAL INTEREST PAID IS  $#######.##"; (NP
     * PMT) + (BAL - AMT)
3400 ON ERROR GOTO 0 :
     PRINT @960,"PRESS 'M' FOR MENU OR '↑' TO CALCULATE NEW A.P.R.";
3420 X$ = INKEY$ :
     IF X$ = ""
         THEN
          3420
3440 IF X$ = "M"
         THEN
          500
3460 IF X$ = "↑"
         THEN
          3020
3480 GOTO 3420
```

Program continued

```
3500 :
     ' ***** DIVIDE BY ZERO ERROR TRAP *****
3520 IF ERR / 2 + 1 = 11
      THEN
       PRINT @ 512,"THE A.P.R. IS LESS THAN ZERO. IT IS IMPOSSIBLE TO
        COMPUTE." :
       RESUME 3400
3540 :
     '
3560 :
     '
4000 :
     ' ***** CALCULATES NUMBER OF PAYMENTS *****
4020 CLS
4040 PRINT "THIS WILL CALCULATE THE NUMBER OF PAYMENTS" :
     PRINT :
     PRINT
4060 PRINT "ENTER MONTHLY PAYMENT  <"PMT">"; :
     INPUT PMT
4080 PRINT "ENTER A.P.R.  <"APR">"; :
     INPUT APR
4100 PRINT "ENTER AMOUNT OF PRINCIPAL  <"AMT">"; :
     INPUT AMT
4120 PRINT "ENTER BALLOON, IF ANY  <"BAL">"; :
     INPUT BAL
4140 PCT = APR / 1200
4160 NP = ( LOG((PMT - PCT * BAL) / (PMT - PCT * AMT))) / ( LOG(1
     + PCT))
4180 IF (NP * 10) - INT(NP * 10) > .49
      THEN
       NP = INT(NP * 10) + 1 :
      ELSE
       NP = INT(NP * 10)
4200 NP = NP / 10
4220 PRINT :
     PRINT :
     PRINT USING "THE NUMBER OF PAYMENTS ARE  ####.#";NP
4240 PRINT :
     PRINT USING "THE TOTAL COST OF THE LOAN IS  $#######.##"; (NP
     * PMT) + BAL
4260 PRINT USING "THE TOTAL INTEREST PAID IS  $#######.##"; (NP
     * PMT) + (BAL - AMT)
4280 PRINT @ 960,"PRESS 'M' FOR MENU , '↑' TO CALCULATE NEW NUMBER OF
      PAYMENTS";
4300 X$ = INKEY$ :
     IF X$ = ""
      THEN
       4300
4320 IF X$ = "M"
      THEN
       500
4340 IF X$ = "↑"
      THEN
       4020
4360 GOTO 4300
4380 :
     '
4400 :
     '
5000 :
     ' ***** CALCULATES 'LOAN PAYOFF' OR 'BALLOON' PAYMENT *****
5020 CLS
5040 PRINT "THIS WILL CALCULATE THE 'LOAN PAYOFF' OR 'BALLOON' PAYMEN
     T":
     PRINT :
     PRINT
5060 PRINT "ENTER MONTHLY PAYMENT  <"PMT">"; :
     INPUT PMT
5080 PRINT "ENTER A.P.R.  <"APR">"; :
     INPUT APR
```

```
5100 IF APR = 0 GOTO 5080
5120 PRINT "ENTER AMOUNT OF PRINCIPAL  <"AMT">"; :
     INPUT AMT
5140 PRINT "ENTER NUMBER OF PAYMENTS  <"NP">"; :
     INPUT NP
5160 IF NP < 1 GOTO 5140
5180 PCT = APR / 1200
5200 BAL = (AMT - (PMT * ((1 - (1 + PCT) ↑ - NP) / PCT))) / ((1
     + PCT) ↑ - NP)
5220 IF (BAL * 100) - INT(BAL * 100) > .49
      THEN
       BAL = INT(BAL * 100) + 1 :
      ELSE
       BAL = INT(BAL * 100)
5240 BAL = BAL / 100
5260 PRINT :
     PRINT USING "THE 'BALLOON' OR 'PAYOFF' PAYMENT IS  $#######.##";
      BAL
5280 PRINT
5300 IF BAL < 0 PRINT USING "THE BANK OWES YOU   $#######.##";
     ABS(BAL) :
     PRINT
5320 PRINT USING "THE TOTAL COST OF THE LOAN IS  $#######.##"; (NP
     * PMT) + BAL
5340 PRINT USING "THE TOTAL INTEREST PAID IS  $#######.##"; (NP
     * PMT) + (BAL - AMT)
5360 PRINT @ 960,"pRESS 'M' FOR MENU, '↑' TO CALCULATE NEW BALLOON OR
      PAYOFF";
5380 X$ = INKEY$ :
     IF X$ = ""
      THEN
       5380
5400 IF X$ = "M"
      THEN
       500
5420 IF X$ = "↑"
      THEN
       5000
5440 GOTO 5380
5460 :
     '
5480 :
     '
6000 :
     ' ***** CALCULATES PRINCIPAL *****
6020 CLS
6040 PRINT "CALCULATES HOW MUCH PRINCIPAL YOU CAN AFFORD"
6060 PRINT :
     PRINT :
     PRINT "ENTER MONTHLY PAYMENT YOU CAN AFFORD  <"PMT">"; :
     INPUT PMT
6080 PRINT "ENTER BEST A.P.R. AVAILABLE  <"APR">"; :
     INPUT APR
6100 IF APR = 0 GOTO 6080
6120 PRINT "ENTER NUMBER OF PAYMENTS YOU ARE WILLING TO MAKE  <"NP">"
     ; :
     INPUT NP
6140 PRINT "ENTER BALLOON, IF ANY  <"BAL">"; :
     INPUT BAL
6160 PCT = APR / 1200
6180 AMT = (PMT * ((1 - (1 + PCT) ↑ - NP) / PCT)) + (BAL * (1
     + PCT) ↑ - NP)
6200 IF (AMT * 100) - INT(AMT * 100) > .49
      THEN
       AMT = INT(AMT * 100) + 1 :
      ELSE
       AMT = INT(AMT * 100)
6220 AMT = AMT / 100
6240 PRINT :
     PRINT :
     PRINT USING "THE AMOUNT OF PRINCIPAL YOU CAN AFFORD IS $#######.
```

Program continued

```
     ##"; AMT
6260 PRINT :
     PRINT USING "THE TOTAL COST OF THE LOAN IS  $#######.##"; (NP
     * PMT) + BAL
6280 PRINT USING "THE TOTAL INTEREST PAID IS  $#######.##"; (NP
     * PMT) + (BAL - AMT)
6300 PRINT @ 960,"PRESS 'M' FOR MENU, '↑' TO CALCULATE A NEW PRINCIPA
     L";
6320 X$ = INKEY$ :
     IF X$ = ""
      THEN
       6320
6340 IF X$ = "M"
      THEN
       500
6360 IF X$ = "↑"
      THEN
       6000
6380 GOTO 6320
6400 :
     '
6420 :
     '
7000 :
     ' ***** CALCULATES END OF YEAR INTEREST FOR INCOME TAX *****
7020 CLS
7040 PRINT "CALCULATES END OF YEAR INTEREST FOR INCOME TAX PURPOSES"
7060 PRINT :
     PRINT :
     PRINT "ENTER MONTHLY PAYMENT  <"PMT">"; :
     INPUT PMT
7080 PRINT "ENTER A.P.R.  <"APR">"; :
     INPUT APR
7100 IF APR = 0 GOTO 7080
7120 PRINT "ENTER AMOUNT OF PRINCIPAL  <"AMT">"; :
     INPUT AMT
7140 PRINT "ENTER NUMBER OF PAYMENTS (WHOLE NUMBERS ONLY)  <"NP">"; :
     INPUT NP
7160 IF NP < 1 OR NP < > INT(NP) GOTO 7140
7180 PRINT "ENTER BALLOON, IF ANY  <"BAL">"; :
     INPUT BAL
7200 PRINT "ENTER NUMBER OF PAYMENTS IN THE FIRST YEAR  <"FY">"; :
     INPUT FY
7220 IF FY < 1 GOTO 7200
7240 PCT = APR / 1200
7260 FY(1) = FY - 1 :
     Y = 0 :
     AMT(1) = AMT :
     YI = 0 :
     Z = 0 :
     MI = 0 :
     YP = 0 :
     MP = 0
7280 FOR X = Y TO FY(1)
7300  MI = PCT * AMT(1)
7320  MP = PMT - MI
7340  AMT(1) = AMT(1) - MP
7360  YI = YI + MI
7380  YP = YP + MP
7400  IF X = NP - 1
       THEN
        XX = 1 :
        GOTO 7460
7420  NEXT
7440 Y = X :
     FY(1) = FY(1) + 12 :
     IF FY(1) = NP - 1
      THEN
       FY(1) = NP
7460 Z = Z + 1
7480 CLS
```

```
7500 IF XX = 1
      THEN
       YP = YP + BAL :
       AMT(1) = AMT(1) - BAL
7520 PRINT "THE INTEREST PAID AT THE END OF YEAR <" Z ; :
     PRINT USING "> IS $######.##"; YI
7540 PRINT "THE PRINCIPAL PAID AT THE END OF YEAR <" Z ; :
     PRINT USING "> IS $######.##"; YP
7560 PRINT "THE PRINCIPAL REMAINING AT THE END OF YEAR <" Z ; :
     PRINT USING "> IS $#######.##"; AMT(1)
7580 IF AMT(1) < 0
      THEN
       PRINT :
       PRINT USING "THE BANK OWES YOU $#######.##"; ABS(AMT(1))
7600 IF XX = 1
      THEN
       XX = 0 :
       GOTO 7720
7620 PRINT :
     PRINT :
     PRINT :
     PRINT "TO CONTINUE PRESS ANY KEY"
7640 PRINT :
     PRINT :
     PRINT "TO ESCAPE THIS ROUTINE PRESS 'E'"
7660 X$ = INKEY$ :
     IF X$ = ""
      THEN
       7660
7680 IF X$ = "E"
      THEN
       7720
7700 YI = 0 :
     YP = 0 :
     GOTO 7280
7720 PRINT @ 960,"PRESS 'M' FOR MENU, '↑' TO CALCULATE NEW YEAR END I
     NTEREST";
7740 X$ = INKEY$ :
     IF X$ = ""
      THEN
       7740
7760 IF X$ = "M"
      THEN
       500
7780 IF X$ = "↑"
      THEN
       7000
7800 GOTO 7740
```

INTERFACE

A Home-Brew Interface

A Home-Brew Interface

by C. R. Vince

After becoming the proud owner of a TRS-80 system in April of 1978, I soon realized that the Level I, while an excellent teaching aid of the BASIC language, left much to be desired when it came to making my computer more than just an expensive toy. After I had waited anxiously for several months (due, I suppose, to the extremely heavy demand), my Level II arrived and was installed.

Now I could really make my "toy" earn its keep, or could I? Yes I could, providing I put out another $439 (Canadian) for an expansion interface. But wait, all that would give me would be a real-time clock, mini-disk controllers, cassette and line printer controllers, and space for an additional PC board. What about my home climate control, model railway control, and other applications? There had to be another way, and I hope that after reading this article you agree with me that there is another way, perhaps even a better way, at least for hobby use.

Introduction

This chapter will describe an interface unit for the TRS-80 Level II that will provide the following features:

1) An interface board to the TRS-80 itself.
2) An output board having up to 16 8-bit parallel output ports.
3) An input board having up to 16 8-bit parallel input ports.
4) A TTY interface board.
5) A home climate control system.
6) A model railroad speed control system.

I would like to point out here that I am no expert in electronic circuit design. In this chapter, most of the circuits have been previously described in other books and publications, including *Microcomputing*. I have merely put them together in one package as simply and as economically as possible. However, the circuits have been tested and do work; in fact, they are in daily use.

To enable novices to understand the workings of the interface unit I have arrowed pertinent lines in the figures to show the directional flow of data on that particular line. The unit has been built on five separate PC boards (excluding the power supply). I use the term PC boards loosely, as these boards were handmade, and only the common lines such as the data bus, the address bus, and the power lines were etched; other lines such as the enable lines were wired.

The edge card connectors used were of the 62-pin type, since they were the least expensive and most readily available at the time; consequently, pin connections given are for the 62-pin variety. Others such as 44 pin could be used, providing they have enough pins to accommodate all lines entering or leaving the board. The edge card connectors were mounted on a piece of wood and like-numbered pins for the +5V, the ground supplies, the data, and the address lines were multiplied from one connector to the next.

I strongly recommend the use of sockets for all ICs, as troubleshooting is made so much easier if you can simply replace a suspect IC to localize the problem. To emphasize this point, when I first plugged in the interface board, I had problems on a new IC, which seemed to work on static bench tests but failed in the unit. By simply exchanging two ICs, the problem was localized in minutes. In addition, don't forget to use .01 uF bypass capacitors on about every fifth IC.

Interface Board

To allow for expansion, I decided to use 74LS367s to buffer all signals coming from the TRS-80 (see Figure 1). I initially buffered the data bus in both directions, but found that this caused problems, because the interface unit bus is, in effect, parallel with the internal TRS-80 bus, and each time an input to the Z-80 processor is effected (e.g., from memory), it also inputs from the interface unit. Since the interface unit data bus had no signal on it, the buffers interpreted this as a high (or a 1). This high caused errors because at times it was transferred to the Z-80, overriding the low (or 0) that should have been there. Therefore, in the final design presented here, the data bus is buffered in one direction only—out to the interface unit. Unfortunately, this results in two data buses in the interface unit: one into the unit (buffered) and one out of the unit (not buffered). This type of arrangement is not uncommon, of course, and presents no problems, except for some additional wiring. Buffered lines are denoted by the "B" following the line designation (e.g., D6B means that data line 6 has been buffered).

The interface board is connected to the TRS-80 by means of a 40-wire cable. At the TRS-80 end an AMP P/N 88103-1 card edge connector (or equivalent) is required. At the interface board end I chose to cable directly to the sockets holding the ICs. This has presented no problems, but connection could be made to the interface board card connector if desired (if enough pins are available). While two 20-wire ribbon cables would seem desirable and easier to connect on the AMP P/N 88103-1, my unit works successfully using regular 40-wire cable about 12 inches long.

The interface board itself can be described in four separate sections:

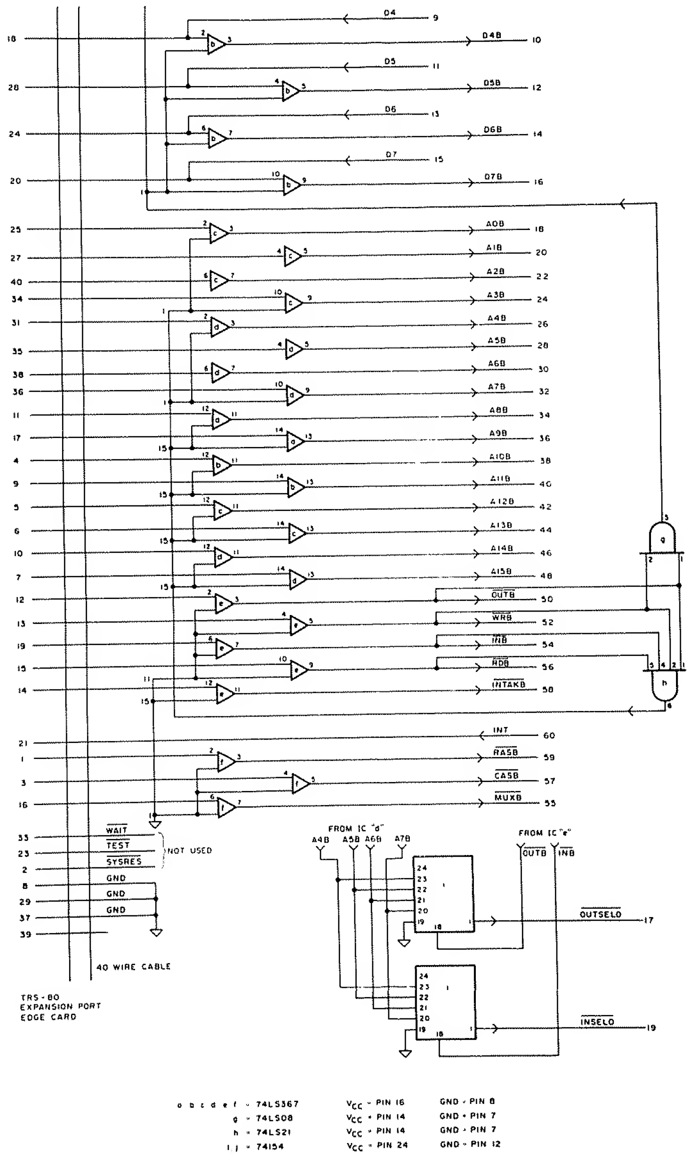


Figure 1. *Interface board*

● *Data Bus Buffers*. The data bus (D0–D7) is buffered by ICs a and b. As mentioned earlier, only data to be output is buffered; input data is presented directly to the TRS-80 without buffering. The buffers are enabled by IC g, which provides a low signal whenever the $\overline{\text{OUTB}}$ or $\overline{\text{WRB}}$ control lines go low. To avoid overloading these buffers, no more than 40 output ports should be used unless additional buffering is provided.

● *Address Bus Buffers*. The address lines (A0–A15) are buffered in much the same way as the data lines. The buffers are enabled by IC h on receipt of a low signal from any one of the four control lines $\overline{\text{OUTB}}$, $\overline{\text{WRB}}$, $\overline{\text{INB}}$, and $\overline{\text{RDB}}$.

● *Control Lines*. The remaining lines from the TRS-80 are what I refer to as control lines. Once again I chose to buffer the control lines that are output from the CPU; the one input line, INT, is not buffered. I decided not to buffer the $\overline{\text{WAIT}}$, $\overline{\text{TEST}}$, and $\overline{\text{SYSRES}}$ lines, since I could foresee no use for them in the near or even distant future, however, I wired them to the interface board just in case, so they could be buffered if desired in the same way as other lines.

The control lines are buffered by ICs e and f. Note that there are three GND lines. These should be connected to the GND of the interface board power supply. The line connected to pin 39 of the TRS-80 edge connector warrants mention here. In the Level I manual (page 228) this line is shown connected to +5 V in the TRS-80. Prior to having my Level II installed, this was the case, however, after the installation of the Level II, I noticed that the trace to pin 39 had been cut and pin 40 had been strapped to pin 39, making pin 39 a ground line. Since I do not know the state of other units with regard to this pin, I recommend that this pin not be wired. Of the nine control lines wired, only two are used by circuits described in this article—the OUT and IN lines—however, the board has been designed to allow for the easy addition of memory and an interrupt board at a later date, which require the additional control lines.

● *Output and Input Port Initial Selectors*. Whenever the TRS-80 executes an input or output (port) instruction, the port address is placed on the lower eight bits of the address bus (A0–A7). At the same time, the $\overline{\text{OUTB}}$ line (on an OUT instruction) or the $\overline{\text{INB}}$ line (on an INP instruction) is enabled. The input or output port initial selectors (IC i or j) are selected by these lines. This causes the high four bits (A4–A7) to be decoded by the selected IC i or j, which are 74154 four-line to 16-line decoders.

The output of the 74154 is used to select a particular input or output board where the final port address is decoded. Thus using this configuration, up to 16 input and 16 output boards could be selected, providing additional buffers are used. In the design presented here only one input and one output board is used, each one containing ports 0 to 15. To select additional boards, simply use the proper output from the 74154s to select the desired board (e.g., to select ports 16 to 31, the output from pin 2 of the relative 74154 would be used).

Output Port Board

The output port board (see Figure 2) provides up to 16 8-bit parallel output ports. In my configuration I have used ports 0 to 8, since this was the physical limit of the size of the board I have available. The board is selected by the $\overline{\text{OUTSEL0}}$ line from the interface board. This enables IC m, a 74154, which now decodes the four bits presented to it on the A0B to A3B lines.

The output from IC m is a low on one of the 16 output lines, corresponding to the binary value of lines A0B to A3B. This low is inverted and subsequently enables a pair of 74LS75 quad latches. The data on the D0B to D7B bus is now latched by the 74LS75 quad latches. The true data is now held by the latch and can be used to control external devices. The use of the edge card connector pins is left to the discretion of the user.

Input Port Board

The input port board (Figure 3) operates in a similar fashion to the output port board. The input board is selected by the $\overline{\text{INSEL0}}$ line from the interface board, enabling the 74154 to decode the final port address, according to the data on the A0B to A3B lines. The output from the 74154 (IC o) strobes the selected input port (IC n), and the data present on the input lines is transferred to the data bus. Again, due to physical limitations, my board only has nine input ports. Either 74LS367s or 74LS368s can be used as the input port; the pinout for either is identical. The only difference is that the 74LS368 inverts the data present on the input lines, whereas the 74LS367 does not. This can be useful.

Imagine a port (x) with 74LS367s and only one input, bit 0, on that port being used. Performing a y = Inp(x) instruction will result in y having a value of 254 or 255, depending on whether the input is high or low. The other seven inputs are seen as high by the TRS-80. However, if 74LS368s are used, then the highs on bits 1 to 7 will be inverted and seen by the TRS-80 as low or 0. Consequently, y will now have a value of 0 or 1. This does make programming a little easier.

Now that we can input and output to the TRS-80, a whole new world has opened up! The following are three of the uses that I have successfully tried to date. Obviously there are many more.

TTY Interface

Shortly after completing the interface board, I purchased a Model 33

Teletype™ at almost bargain-basement price from a local dealer at a clearance sale. I constructed the TTY board (see Figure 4) in a couple of evenings. It uses a popular UART, the AY-3-1015, and was selected primarily because

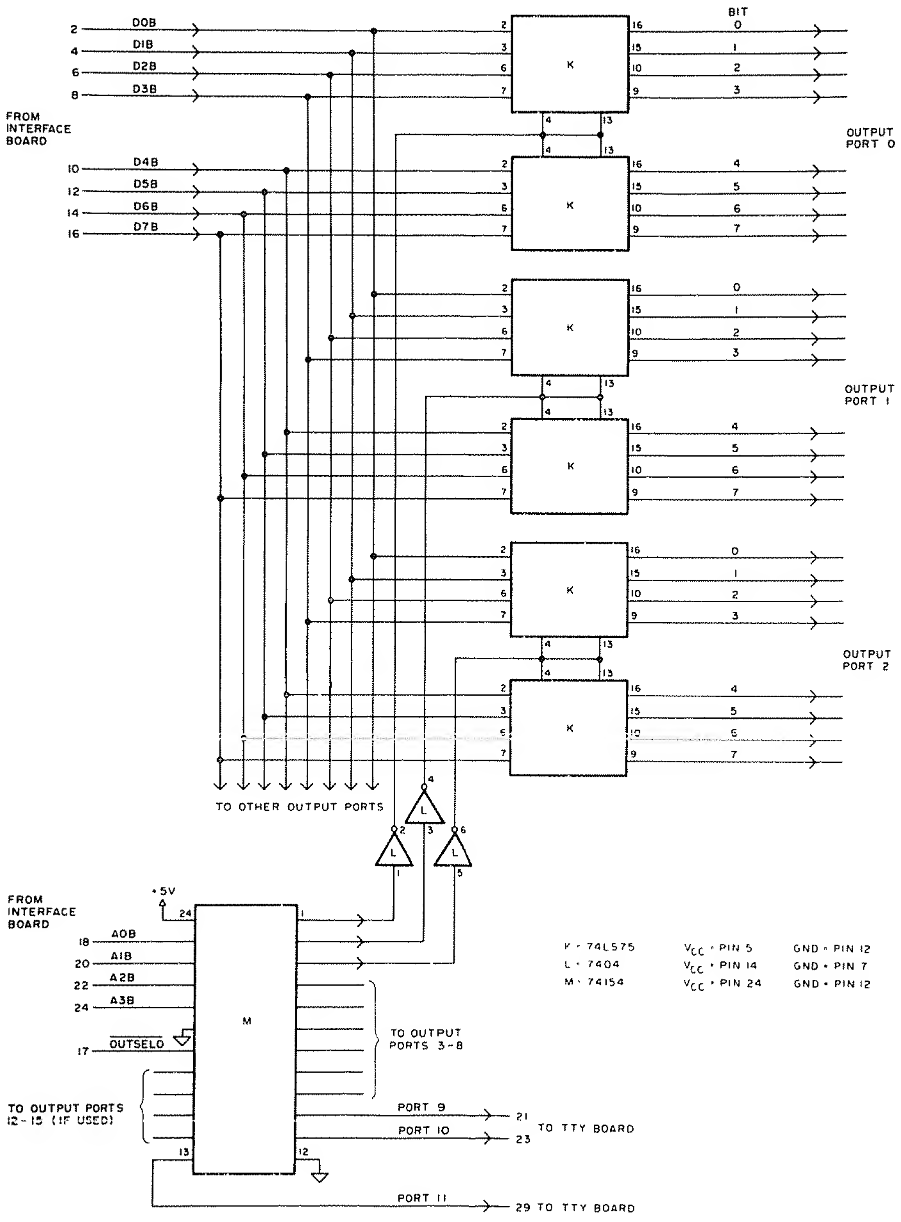


Figure 2. *Eight-bit parallel output board (only three ports shown)*

of the single 5 V supply required. Other UARTs would probably work just as well. Whichever UART you purchase, I suggest you obtain a copy of the specification sheets, as many variations are allowed (e.g., parity, number of stop bits, number of bits/character, etc).

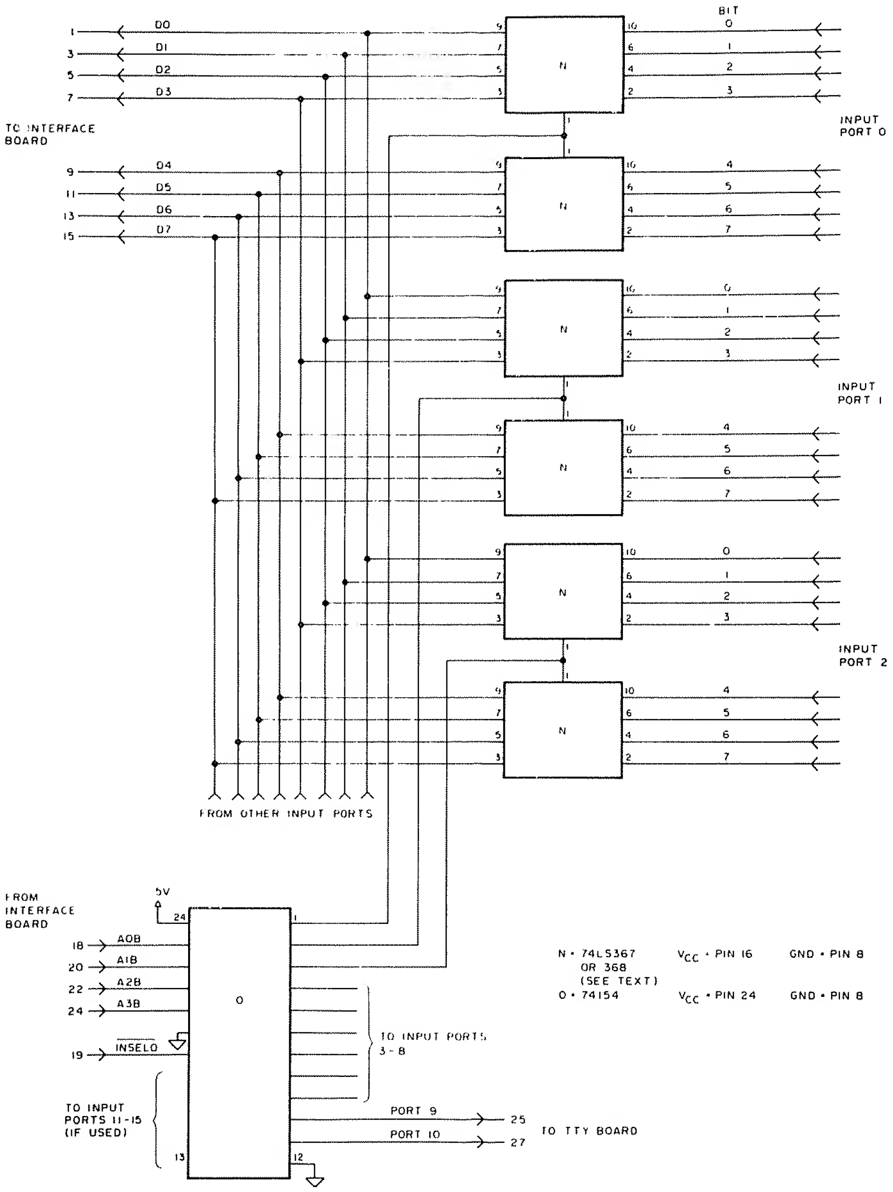
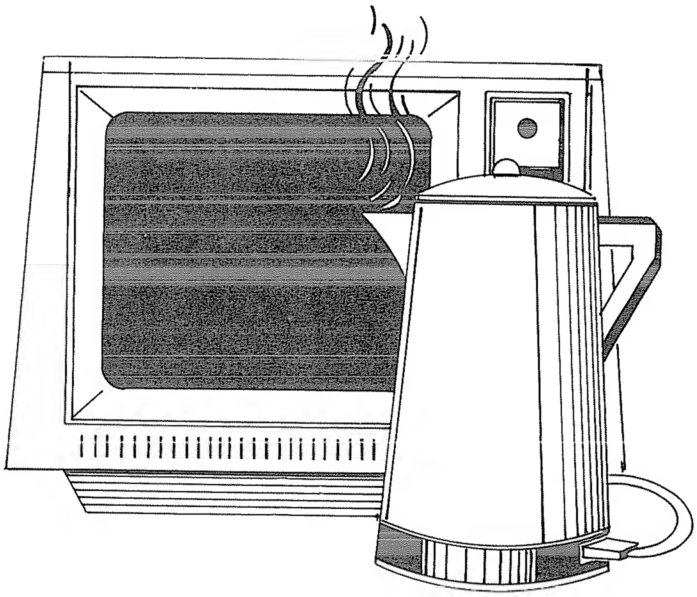


Figure 3. *Eight-bit parallel input board (only three ports shown)*

To list the numerous variations here would be too lengthy, however, the circuit as shown will run a Model 33 Teletype™ at 110 baud, 20 mA current loop in half duplex operation. No programming is necessary to convert the serial data to parallel or vice versa, as this is done by the UART. The 555 timer circuit supplies clock pulses at 16 times the desired baud rate, therefore, the clock frequency for 110 baud is 1760 Hz. The actual serial data is transmitted to and received from the TTY by the two 4N26 optical couplers and the 2N2222 transistors. These couplers provide electrical isolation between the TTY and UART.

At the start of any program that will input or output data through the UART, an OUT 11,0 instruction should be used to reset all internal UART registers and flags to 0. To input data from the TTY, an INP9, (x) instruction will enable the $\overline{\text{SWE}}$-0 line (status word enable). If bit 1 is a 1, the DAV (data available flag) line will be high, indicating that the UART does, in fact, have data to input. An INPIO(x) will enable the $\overline{\text{RDE}}$ line (received data enable) and will result in the data being placed onto the data bus.

Following this, an OUT 10,0 should be executed to enable the $\overline{\text{RDAV}}$ line (reset data available flag). Obviously, to ensure that no input is missed, these instructions should be contained in a loop, with a branch out only when a character is read. To output data to the TTY an INP9,(x) instruction will again result in the status word being output on the data bus. This time, however, we are interested in bit 0, which will be the TBMT



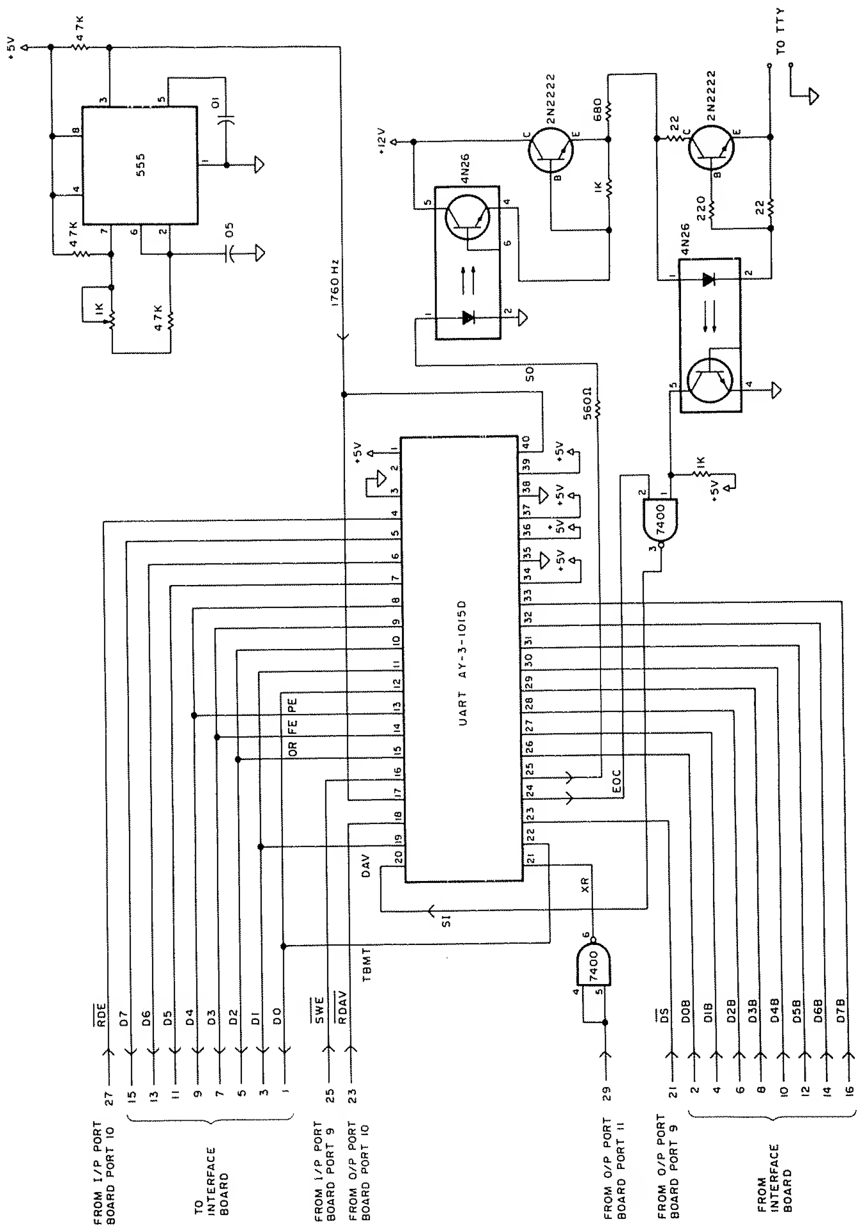


Figure 4. *TTY interface*

flag (transmitter buffer empty). If the TBMT flag is a 1 then the data may be output to the UART for transmission. To do this, an OUT9,(x) instruction is required.

Note that during transmission from the UART, the EOC line on pin 24 goes low. This keeps the output from the 7400 high, which prevents the UART from seeing the transmitted character on the receiver side (pin 20). Three other flags are output on the data bus: OR (overrun), FE (framing error), and PE (parity error). These can be checked by software if required, but this is not absolutely necessary.

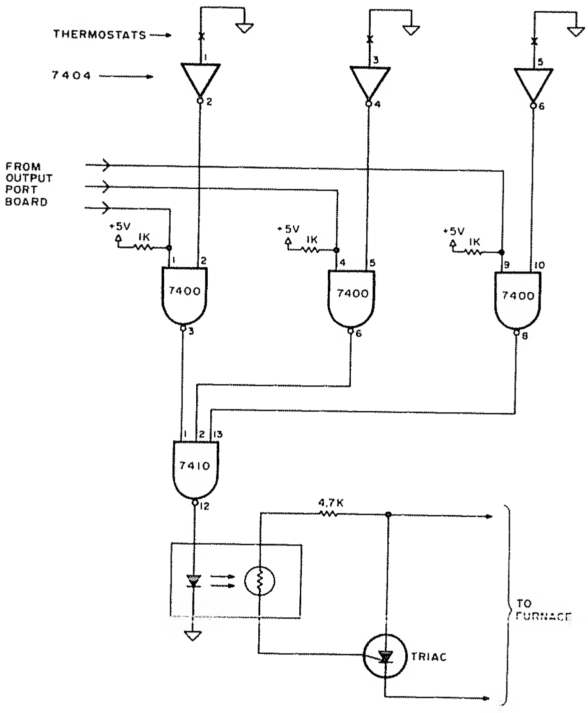


Figure 5. *Furnace control*

Home Climate Controls

Programmable timers that will turn down the thermostat setting at night are available, however, the cost of two more thermostats is even less, and besides that, it gives your TRS-80 something to do while you are working! My house has three levels, so a thermostat on each level is enabled by a signal from an output port under program control. As a safety feature, I have wired three outputs through a plug and socket arrangement, so that in the event of a failure of the computer, by disconnecting the plug, all three thermostats are automatically enabled, as shown in Figure 5. (There's nothing worse than trying to fix a program bug when you can't see the monitor for the ice crystals!)

The triac—I used one from my junk box, as the voltage and current demands are minimal (check this on your unit)—turns on the furnace as the thermostat used to do. The triac is turned on by a simple circuit consisting of an LED and an LDR (light dependent resistor), which I bought at Radio Shack. Of course, these two items must be enclosed in a lightproof container to be effective.

To provide a means of keeping time in the computer, I used a one-minute pulse from a digital clock (which I had built some time ago) connected to input port 0. The clock itself is driven by a 160 kHz crystal, which is divided by a number of binary counters (7493s) connected in series to produce a one-minute pulse. The input port is continuously monitored for a change in state. Other methods could be used, e.g., a FOR-NEXT loop or a 555 timer circuit if you are not too concerned about accuracy.

Model Railway Speed Control

The speed control shown in Figure 6 is a simple digital-to-analog converter circuit. With bit 3 low, the output of the converter circuit is low, hence Q1 and Q2 are turned off. With bit 3 high, a voltage is presented to the base of Q1, turning it and Q2 on. The exact voltage is determined by the binary value of bits 0, 1, and 2. The output voltage appearing at the emitter of Q2 is incremented in eight steps by decrementing the binary value of the four inputs to the 7406 (bits 0–3).

Perhaps the easiest way to explain this is by saying that with a value of 8, Q1 and Q2 are off and with a value of 0, they are full on. Thus, the train is stopped with a value of 8 and runs at its fastest speed with a value

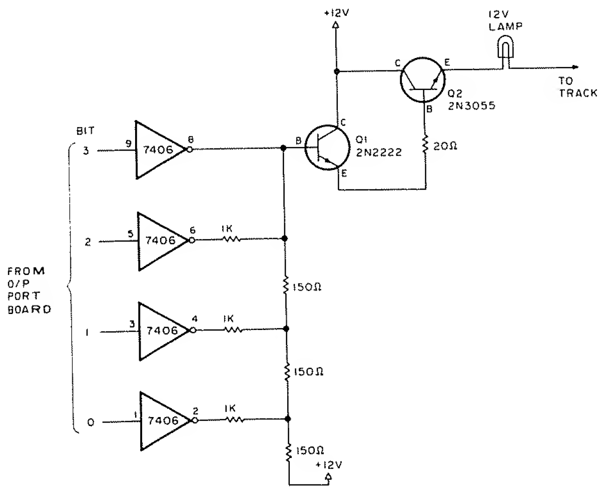


Figure 6. *Model railway speed control*

of 0 presented to the converter circuit from the output port. For values between 0 and 7, the train runs at a correspondingly slower speed. The actual voltage is from about 6 V, which is the lowest voltage that most HO-scale trains will run at, to about 11.5 V (assuming a 12 V supply).

Software

If you decide to build the TTY interface board, the following programs should greatly enhance the capabilities of your computer. Program Listings 1 and 2 allow the use of the resident TRS-80 LLIST and LPRINT commands with a Model 33 (or similar) TTY and the TTY interface board previously described.

The TRS-80 is designed to produce hard copy on a line printer through a memory mapped I/O port at address 14312 (37E8H). The software routines necessary to permit this function are continued within the BASIC ROM.

I first thought that I would be able to use these routines by decoding address 14312 and wiring the UART circuit to it. However, I found that the ROM routines do not issue a line feed command; at the end of a line of print only a carriage return command is issued. Obviously the Radio Shack line printer automatically line feeds whenever it receives a carriage return. A 33 TTY does not!

With the help of the RSM monitor, I eventually found the answer. On power-up initialization a number of addresses in RAM are loaded with information used by the BASIC interpreter. Two of these addresses, 16422 and 16423 (4026H & 4027H), are loaded with the entry point of the line printer output routine—1421 (0580H). By providing my own TTY handling routine and directing the BASIC interpreter to it by changing the contents of 16422 and 16423, the TRS-80 can output to the TTY rather than to the line printer.

Program Listings 1 and 2 do just that. Program Listing 1 is the actual assembly language program which I produced using the Radio Shack Editor/Assembler. It generates a line feed whenever a carriage return is performed. It also generates a CR and LF when 64 characters are printed on any line; thus the hard copy looks exactly the same as displayed on the monitor.

If you have the Editor/Assmbler program, I recommend producing Program Listing 1 and making a tape copy of it. Simply load it using the system command and enter a "/". This loads the pointer addresses 16422 and 16423 and returns to BASIC.

For those who do not have the Editor/Assembler, Program Listings 2 and 3 are provided. These are BASIC language programs which POKE the machine-language program into high memory. Once POKEd, the

BASIC programs can be deleted, and the TTY handler program will remain in high memory until power is removed. Program Listing 2 is for 16K and Program Listing 3 is for 4K. Remember that whatever method you use, the memory size must be set to 20224 for a 4K computer or 32512 for a 16K computer. If you use Program Listings 2 or 3 take care when entering the DATA statements. One wrong entry will probably cause your computer to get lost, which will require a power-off reset to get it back, which will erase your program entirely.

Program Listing 4 is a TTY test and demonstration program. It initially requests the operator to input the number of "fox" messages required and then goes on to output the standard TTY test message: "The quick brown fox jumps over the lazy dog. 0123456789". The operator is then prompted to type a message. Note that the message is terminated with a semicolon (;). The typed letters are displayed on the monitor screen and are also typed back on the TTY, providing that no error flags are set.

Thus, if you have a suspect TTY, the location of the problem can be determined (i.e., keyboard or printing unit) by using this program. For example, if the "fox" message types OK and the characters displayed on the monitor are incorrect, then obviously the trouble is in the keyboard or transmitter portion of the TTY. Of course, the UART wiring is also checked by this test.

Line 1090 is part of a continuous loop monitoring the status word flags for a change. If a change in state on any flag except the TBMT flag is detected, line 1100 will determine whether an error is present in the received character. If an error exists, then a transfer will be made to line 1150, where the particular error is determined. If no error has been detected by the UART, control will drop through to line 1110, where the received character is processed. To save typing and memory, lines 1150-1195 can be replaced by:

```
1150 A$ = "TIY ERROR, FAULT CODE":S$ = STR$(S):A$=A$+S$
1160 PRINTA$:GOSUB1300
```

In this case, you must break down the decimal fault code given into binary and Table 1 is used to determine the error.

Bit	Meaning
0	Always 0 (TBMT)
1 = 1	DAV (data available)
2 = 1	OV (overrun error)
3 = 1	FE (framing error)
4 = 1	PE (parity error)

Table 1

Conclusion

In addition to the uses already described, the interface unit has been used to turn on and off outside lighting at Christmastime and as a telephone dialer. It is presently being used to control basement lighting, in addition to the climate control system previously described.

Providing care and patience are used, even a novice should be able to build this unit, as no special tools are required. Once this unit is built, I am sure that you will discover that your TRS-80 is no longer just an "expensive toy," but rather a useful addition to your household.

Encyclopedia Loader

Program Listing 1

```
                 ØØØØ5 ;**************** TTYOP1 ****************
                 ØØØ1Ø ;TTYOP1 ALLOWS USE OF A REGULAR TTY WITH THE
                 ØØØ2Ø ;TRS-8Ø. THIS ALLOWS DIRECT USE OF LLIST AND
                 ØØØ3Ø ;LPRINT COMMANDS.  IT RESIDES AT 7FØØ(H) WHICH
                 ØØØ4Ø ;IS THE ADDRESS (32512D) THAT MUST BE ANSWERED
                 ØØØ5Ø ;IN RESPONSE TO "MEMORY SIZE?".  AFTER LOADING
                 ØØØ6Ø ;A "/" WILL LOAD THE DCB (4026H & 4027H) WITH
                 ØØØ7Ø ;THE "START" ADDRESS AND WILL THEN JUMP TO
                 ØØØ8Ø ;BASIC
                 ØØØ9Ø
7FØØ             ØØ1ØØ         ORG     7FØØH
7FØØ D3ØB        ØØ11Ø         OUT     (11),A         ;RESET UART
7FØ2 211Ø7F      ØØ12Ø         LD      HL,START       ;ADDR OF TTYOP1
7FØ5 22264Ø      ØØ13Ø         LD      (4Ø26H),HL     ;INTO DCB
7FØ8 212A4Ø      ØØ14Ø         LD      HL,4Ø2AH       ;CHAR COUNT ADDR
7FØB 364Ø        ØØ15Ø         LD      (HL),64        ;LOAD # OF CHAR/LINE
7FØD C3191A      ØØ16Ø         JP      1A19H          ;JP TO BASIC
7F1Ø 79          ØØ17Ø START   LD      A,C            ;CHAR TO BE O/P
7F11 FEØD        ØØ18Ø         CP      13             ;CK IF CR
7F13 2ØØ4        ØØ19Ø         JR      NZ,A1          ;JP IF NOT
7F15 CD2F7F      ØØ2ØØ         CALL    CRLF           ;CR+LF ROUTINE
7F18 C9          ØØ21Ø         RET
7F19 CD267F      ØØ22Ø A1      CALL    OPCHAR         ;CHAR O/P ROUTINE
7F1C DD7EØ5      ØØ23Ø         LD      A,(IX+5)       ;LD # OF CHAR LEFT
7F1F FEØØ        ØØ24Ø         CP      Ø              ;CK IF CRLF NEEDED
7F21 CØ          ØØ25Ø         RET     NZ             ;RET IF NOT
7F22 CD2F7F      ØØ26Ø         CALL    CRLF           ;CR+LF ROUTINE
7F25 C9          ØØ27Ø         RET
7F26 CD3E7F      ØØ28Ø OPCHAR  CALL    CKTBMT         ;CK IF TBMT
7F29 D3Ø9        ØØ29Ø         OUT     (Ø9),A         ;O/P CHAR
7F2B DD35Ø5      ØØ3ØØ         DEC     (IX+5)         ;DEC CHAR COUNTER
7F2E C9          ØØ31Ø         RET
7F2F 3EØD        ØØ32Ø CRLF    LD      A,13           ;LOAD CR
7F31 CD267F      ØØ33Ø         CALL    OPCHAR         ;O/P CHAR
7F34 3EØA        ØØ34Ø         LD      A,1Ø           ;LOAD LF
7F36 CD267F      ØØ35Ø         CALL    OPCHAR         ;O/P LF
7F39 DD36Ø54Ø    ØØ36Ø         LD      (IX+5),64      ;RELOAD CHAR COUNTER
7F3D C9          ØØ37Ø         RET
7F3E F5          ØØ38Ø CKTBMT  PUSH    AF             ;SAVE CHAR IN A
7F3F DBØ9        ØØ39Ø         IN      A,(Ø9)         ;I/P UART FLAGS
7F41 E6Ø1        ØØ4ØØ         AND     1              ;STRIP OFF TBMT
7F43 FEØ1        ØØ41Ø         CP      1              ;CK IF MT
7F45 2ØF8        ØØ42Ø         JR      NZ,CKTBMT+1    ;JP IF NOT
7F47 F1          ØØ43Ø         POP     AF             ;RESTORE A REG
7F48 C9          ØØ44Ø         RET
7FØØ             ØØ45Ø         END     7FØØH
ØØØØØ TOTAL ERRORS
```

Program Listing 2

```
 1Ø REM  THIS IS THE 16K VERSION OF A BASIC PROGRAM FOR LOADING
 2Ø REM  A MACHINE LANGUAGE PROGRAM INTO HIGH MEMORY TO ALLOW
 3Ø REM  USE OF A TTY WITH LLIST AND lPRINT COMMANDS.
 4Ø REM  ONCE LOADED, THE TTY HANDLER WILL REMAIN IN MEMORY
 5Ø REM  UNTIL POWER IS REMOVED, AND THIS PROGRAM MAY BE ERASED.
 6Ø REM  MEMORY SIZE MUST BE SET AT 32512 PRIOR TO RUNNING THIS
 7Ø REM  PROGRAM. OUTPUT TO THE TTY IS THROUGH PORT 9.
 9Ø CLS
1ØØ FOR X = 32512 TO 32584
11Ø   READ Y:
      POKE X,Y
12Ø   NEXT
```

Program continued

```
 130 POKE 16526,0:
     POKE 16527,127
 135 PRINT "TTY HANDLER LOADED"
 140 X = USR(0)
 150 END
1000 DATA 211,11,33,16,127,34,38,64,33,42,64,54,64,195,25,26,121
1010 DATA 254,13,32,4,205,47,127,201,205,38,127,221,126,5,254,0
1020 DATA A192,205,47,127,201,205,62,127,211,9,221,53,5,201,62,13
1030 DATA 205,38,127,62,10,205,38,127,221,54,5,64,201,245,219,9
1040 DATA 230,1,254,1,32,248,241,201
```

Program Listing 3

```
  10 REM  THIS IS THE 4K VERSION OF A BASIC PROGRAM FOR LOADING
  20 REM  A MACHINE LANGUAGE PROGRAM INTO HIGH MEMORY TO ALLOW
  30 REM  USE OF A TTY WITH LLIST AND lPRINT COMMANDS.
  40 REM  ONCE LOADED, THE TTY HANDLER WILL REMAIN IN MEMORY
  50 REM  UNTIL POWER IS REMOVED, AND THIS PROGRAM MAY BE ERASED.
  60 REM  MEMORY SIZE MUST BE SET AT 20224 PRIOR TO RUNNING THIS
  70 REM  PROGRAM. OUTPUT TO THE TTY IS THROUGH PORT 9.
  90 CLS
 100 FOR X = 20224 TO 20296
 110  READ Y:
      POKE X,Y
 120  NEXT
 130 POKE 16526,0:
     POKE 16527,79
 135 PRINT "TTY HANDLER LOADED"
 140 X = USR(0)
 150 END
1000 DATA 211,11,33,16,79,34,38,64,33,42,64,54,64,195,25,26,121
1010 DATA 254,13,32,4,205,47,79,201,205,38,79,221,126,5,254,0
1020 DATA 192,205,47,79,201,205,62,79,211,9,221,53,5,201,62,13
1030 DATA 205,38,79,62,10,205,38,79,221,54,5,64,201,245,219,9
1040 DATA 230,1,254,1,32,248,241,201
```

Program Listing 4

```
1000 CLEAR 500
1010 INPUT "# OF FOX MESSAGES";K
1020 OUT 11,0:
     GOSUB 1300
1023 A$ = "TTY TEST PROGRAM":
     GOSUB 1300
1026 A$ = "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.0123456789"
1030 FOR Z = 1 TO K
1035  GOSUB 1300
1040  NEXT
1050 A$ = "PLEASE TYPE A MESSAGE":
     GOSUB 1300
1070 CLS
1080 A$ = ""
1090 S = INP(9):
     S = S AND 30:
     IF S = 0
      THEN
       1090
1100 IF S > 2
      THEN
       1150
1110 A = INP(10):
     OUT 10,0:
     IF A = 59
      THEN
       1130:
       REM  ; TERMINATES INPUT
1120 X$ = CHR$(A):
     A$ = A$ + X$:
     PRINT @256,A$:
     GOTO 1090
1130 GOSUB 1330
1131 GOSUB 1300
1135 A$ = "TTY LOOKS OK--PLEASE TYPE AGAIN":
     GOSUB 1300
1137 GOTO 1070
1150 SS = S:
     SS = SS AND 16:
     IF SS < > 0
      THEN
       1180
1160 SS = S:
     SS = SS AND 8:
     IF SS < > 0
      THEN
       1190
1170 SS = S:
     SS = SS AND 4:
     IF SS < > 0
      THEN
       1195
1171 GOTO 1197
1180 A$ = "PARITY ERROR":
     PRINT A$:
     GOSUB 1300
1182 GOTO 1160
1190 A$ = "FRAMING ERROR":
     PRINT A$:
     GOSUB 1300
1192 GOTO 1170
1195 A$ = "OVERRUN ERROR":
     PRINT A$:
     GOSUB 1300
1197 A$ = "TYPE AGAIN":
     PRINT A$:
     GOSUB 1300
1200 FOR Z = 1 TO 100:
      NEXT
1210 OUT 11,0:
     GOTO 1080
1300 FOR X = 1 TO LEN(A$)
1310  C$ = MID$(A$,X,1):
      C = ASC(C$):
      GOSUB 1360
1320  NEXT
1330 C = 13:
     GOSUB 1360
1340 C = 10:
     GOSUB 1360
1350 RETURN
1360 S = INP(9):
     S = S AND 1:
     IF S = 0
      THEN
       1360
1370 OUT 9,C:
     RETURN
```

TUTORIAL

A Handle On Programming: Store and Recall

by Allan S. Joffe W3KBM

This is an effort to show your TRS-80's ability to store the contents of the screen and recall those contents on demand. The screen contents may be text or graphics, single or in combination. Program Listing 1 shows the initial approach. The program uses a combination of BASIC and machine language called from BASIC by means of the USR(0) function.

While the listing is reasonably well commented, a bit of explanation may be in order. Lines 5 to 140 are involved in storing the information that line 10 writes to the screen. Line 110 gives the screen limits in decimal; a full screen is 1024 bytes or 1K. Line 100 contains the decimal value of memory location 5000 hex. This value increases by one each time the POKEing loop cycles. You will notice that it takes a while for the screen contents to be POKEd into memory. When the POKEing is completed, and the delay loop in line 160 runs out, USR(0) gets the information that is stored starting at memory location 5000 hex and block transfers it to the screen so that the entire printout springs to life on the screen.

The block transfer routine that USR(0) calls is located in memory at 7000 hex or 28672 decimal. The machine-language code is as follows:

11 00 3C	Destination for data in memory (video screen)
21 00 50	Location of source of data to be moved
01 00 04	Number of bytes to be moved (1024)
ED B0 C9	Execute block transfer and return to program

The two USR(0) pointers 408E and 408F have to contain 00 and 70 respectively so that USR(0) will go where the machine code is stored starting at location 7000 hex.

The main drawback to this program is the time the POKEing section takes to put the screen contents into memory. There is a way to speed up execution. Examine Program Listing 2 and you will see that the POKE routine in BASIC has vanished. In its place is another use of the USR(0) routine which accomplishes the same thing with greater speed. Line 20 reverses the USR(0) function from its original purpose shown in Program Listing 1. In Program Listing 1, USR(0) recovers the information stored in memory and brings it to the screen. Line 20 of Program Listing 2 reverses this function so that the routine stores the screen information into memory. Once the information is stored, in order to get it back to the screen, we reverse the USR(0) routine again using line 70. This time, USR(0) retrieves the information from memory and displays it on the video screen. Under control of your BASIC program, you have modified USR(0) by reversing the source and the destination codes in the machine-language program using the POKE commands in lines 20 and 70.

A logical question is how to store more than one screen's worth of information. To store a full screen (let's call it a page), you need to allow room in memory for 1K, or 1024 bytes. This means that successive pages must be spaced at 1K intervals. With this in mind, let's stick with the first page address of 5000 hex or 20480 decimal. The next two pages would have addreses of 5400 hex or 21504 decimal and 5800 hex or 22528 decimal. Three pages will do nicely for purposes of demonstration, as accomplished in Program Listing 3.

POKE Location	Value to POKE
16526	0
16527	112
28672	17
28673	0
28674	60
28675	33
28676	0
28677	88
28678	1
28679	0
28680	4
28681	237
28682	176
28683	201

Table 1. *POKE locations and values for use without a monitor program*

Notice that Program Listing 3 uses the INPUT statement. This time around you are going to store text. Please note that if your stored text includes commas or colons the first character you enter should be a quotation mark ("). If you do not do this for text including the comma or the colon, you will get an EXTRAS IGNORED error message.

We are still using our same machine code and the technique of making the USR(0) routine store and then recall screen information. The programming steps are simple and direct. We could use various subroutine methods of constructing the program, but they can complicate the idea of program flow.

Lines 30, 60, and 90 serve as page select lines so that the pages of text are stored in consecutive 1K sections of memory. Lines 60 and 90 differ from line 30 in form. This is because, once having POKEd 28677,60 for the first time, you need not repeat this code. The only code that changes with succeeding lines is the POKE location 28674 which controls the page locations in memory. You will notice the same thing in the portion of the program that extracts the text from memory and puts it on the video screen. Line 225 with its attendant REM simply removes the INPUT ? from the screen when the text is retrieved. It prints CHR$(32), a space which washes out the INPUT ?. If this did not happen, your text would be preceded by a question mark when you recalled it to the screen. You can eliminate the quotation mark by using the same method should you so desire.

Program Listing 4 is self-contained to produce and store three different graphics displays which you can then recall. Line 20 fills the screen with numbers. Line 50 puts a solid graphics block in the upper left-hand quadrant of the screen. Line 80 puts a larger, dotted graphics display on the screen. The fundamentals of the program are otherwise unchanged.

The number of pages you can store depends on the size of your available memory and on how large your BASIC programs are going to be for each page. BASIC starts its programming memory usage at location 17129 (42E9 hex). Our first page is located at 20480 (5000 hex); so there is room to program in BASIC. If you stay within these bounds, you can easily store seven pages in a 16K system.

This system of store and recall could be used to implement simple animation. If your animation is confined to a small area of the screen, say the top half, you can easily double the number of pages stored by cutting the bytes transferred in half. To do this, you would alter the machine-code segment, 01 00 04 to 01 00 02, to store and transfer just the top half of the screen. In other words, you split the screen horizontally. While it is beyond the scope of these programs to split the screen vertically, these are the first steps to doing so.

Use T-BUG or some other monitor to get the initial program of machine code into the locations beginning with 7000 hex. If you are using T-BUG in the original, load in the machine code; don't forget that you must change the USR(0) locations of 408E and 408F to their proper values. When you enter your BASIC program, T-BUG will be overwritten, but this will not matter. If you are fortunate enough to have a relocated copy of T-BUG that resides in high memory, then T-BUG will stay resident in memory. This is a convenience, but is not necessary to explore the programs that I have presented here. Table 1 contains the POKE locations and values required for use without a monitor program.

Program Listing 1

```
  1 REM  LISTING #-1 STORE AND RECALL
  2 REM  USR(0) ROUTINE STORED BEGINNING WITH HEX LOCATION 7000WHICH
      IS DECIMAL 28672.  MACHINE CODE    11 00 3C 21 00 50 0100 04 ED
      B0 C9 ....LOCATION 408E(HEX) IS 00...LOCATION 408F(HEX)IS 70...
      .
  4 REM  THIS PART OF THE PROGRAM FILLS SCREEN WITH NUMBERS ANDSTORE
    S THE SCREEN MATERIAL IN MEMORY STARTING AT LOCATION 5000H
  5 CLS
 10 FOR X = 1 TO 200:
      PRINT X;:
      NEXT X
100 Y = 20479
110 FOR X = 15360 TO 16383
115  Y = Y + 1
120  A = PEEK(X)
130  POKE Y,A
140  NEXT X
150 CLS
160 FOR T = 1 TO 1000:
      NEXT T
165 REM  NOW USR(0) READS STORED INFORMATION BACK TO SCREEN
170 J = USR(0)
180 GOTO 180
```

Program Listing 2

```
  1 REM  LISTING #-2 STORE AND RECALL
  2 REM  USR(0) ROUTINE STORED BEGINNING WITH HEX LOCATION 7000WHICH
      IS DECIMAL 28672.  MACHINE CODE    11 00 3C 21 00 50 0100 04 ED
      B0 C9 ....LOCATION 408E(HEX) IS 00...LOCATION 408F(HEX)IS 70...
      .
  4 REM  THIS PART OF THE PROGRAM FILLS SCREEN WITH NUMBERS ANDSTORE
    S THE SCREEN MATERIAL IN MEMORY STARTING AT LOCATION 5000H
  5 CLS
 10 FOR X = 1 TO 200:
      PRINT X;:
      NEXT X
 20 POKE 28674,80:
    POKE 28677,60
 30 REM  LINE 20 PUTS SCREEN CONTENTS INTO MEMORY STARTING ATLOCATIO
    N 5000 HEX
 40 FOR T = 1 TO 1000:
      NEXT T
 50 J = USR(0)
 60 CLS
 70 POKE 28674,60:
    POKE 28677,80
 80 REM  LINE 70 GETS STORED MEMORY CONTENTS AND PUTS THEM TO THESCR
    EEN
 90 J = USR(0)
150 CLS
160 FOR T = 1 TO 1000:
      NEXT T
165 REM  NOW USR (0) READS STORED INFORMATION BACK TO SCREEN
170 J = USR(0)
180 GOTO 180
```

Program Listing 3

```
  0 REM  PROGRAM LISTING #-3 STORE AND RECALL
  1 REM  THIS PART OF PROGRAM ALLOWS YOU TO FILL THREE FRAMES OFMEMO
    RY.  MEMORY PAGE LOCATIONS  #-1 5000 HEX(20480 DECIMAL)#-2 5400
    HEX(21504 DECIMAL)  #-3 5800 HEX(22528 DECIMAL)
  2 REM  FOR USR(0) ROUTINE LOCATION 408E IS 00 AND LOCATION 408FIS
    70.  THESE NOTATIONS ARE IN HEX
 10 CLS
 15 CLEAR 300
 20 INPUT A$
 30 POKE 28674,80:
    POKE 28677,60
 40 J = USR(0)
 50 CLS :
    INPUT B$
 60 POKE 28674,84
 70 J = USR(0)
 80 CLS :
    INPUT C$
 90 POKE 28674,88
100 J = USR(0)
110 REM  THIS PART OF THE PROGRAM ALLOWS RECALL OF THE MEMORY PAGES
    IN THE ENTERED SEQUENCE.
120 CLS
200 INPUT "PRESS ENTER";Z
210 POKE 28674,60:
    POKE 28677,80
220 J = USR(0)
225 PRINT @0, CHR$(32):
    REM  THIS GETS RID OF ? ON RECALL TO SCREEN
230 INPUT "PRESS ENTER";Z
240 POKE 28677,84
250 J = USR(0)
255 PRINT @0, CHR$(32)
260 INPUT "PRESS ENTER";Z
270 POKE 28677,88
280 J = USR(0)
285 PRINT @0, CHR$(32)
300 END
```

Program Listing 4

```
  0 REM  LISTING #-4 STORE AND RECALL
  1 REM  THIS PART OF PROGRAM ALLOWS YOU TO FILL THREE FRAMES OFMEMO
    RY.  MEMORY PAGE LOCATIONS  #-1 5000 HEX(20480 DECIMAL)#-2 5400
    HEX(21504 DECIMAL)  #-3 5800 HEX(22528 DECIMAL)
  2 REM  FOR USR(0) ROUTINE LOCATION 408E IS 00 AND LOCATION 408FIS
    70.  THESE NOTATIONS ARE IN HEX
  3 REM  DIFFERENT FORM OF ENTERING INFORMATION TO DEMONSTRATE HOWYO
    U COULD PUT GRAPHICS PROGRAMS INTO THE STORE FRAMES
 10 CLS
 15 CLEAR 300
 20 FOR X = 1 TO 100:
     PRINT X;:
     NEXT X
 30 POKE 28674,80:
    POKE 28677,60
 40 J = USR(0)
 50 CLS :
    FOR X = 1 TO 20:
     FOR Y = 1 TO 10:
```

Program continued

```
     SET(X,Y):
     NEXT Y,X
 60  POKE 28674,84
 70  J = USR(0)
 80  CLS :
     FOR X = 1 TO 50 STEP 2:
      FOR Y = 1 TO 30 STEP 2:
       SET(X,Y):
       NEXT Y,X
 90  POKE 28674,88
100  J = USR(0)
110  REM  THIS PART OF THE PROGRAM ALLOWS RECALL OF THE MEMORY PAGE
     S IN THE ENTERED SEQUENCE.
120  CLS
200  INPUT "PRESS ENTER";Z
210  POKE 28674,60:
     POKE 28677,80
220  J = USR(0)
225  FOR T = 1 TO 1000:
      NEXT T
230  CLS :
     INPUT "PRESS ENTER";Z
240  POKE 28677,84
250  J = USR(0)
255  FOR T = 1 TO 1000:
      NEXT T
260  CLS :
     INPUT "PRESS ENTER";Z
270  POKE 28677,88
280  J = USR(0)
290  FOR T = 1 TO 1000:
      NEXT T
300  END
```

Prime Up Your 80

by Jim Mellander

Tired of hobbling along at BASIC speeds? Do you want to fully exploit the speed and power of your TRS-80? If the answer is yes, drop your BASIC and jump on the assembly-language bandwagon!

The Problem

To get this show on the road, we need to select a problem to program that is challenging, yet simple. Let's try to calculate prime numbers. A prime number is a number which will divide evenly by only itself and one. The first eight prime numbers are: 3,5,7,11,13,17,19, and 23. Before attempting to write this program, we should consider the problem in some detail. First, we notice that only odd numbers need to be considered since all even numbers can be divided by two. This cuts in half the amount of numbers to check. How are we going to determine if a number is prime? We could divide the prospective prime by successive odd numbers. If the division comes out even, we know the number is not prime and could go on and check the next odd number. If the prospective prime survives all those divisions and none comes out even, then the number must be prime—because that is the definition of a prime number.

Is our algorithm complete? Not quite. We haven't specified how the computer is to know when it has finished dividing. The first thought is to check all the odd numbers up to the number we are testing. That method will work, but it involves far more divisions than are necessary, and dividing is the slowest of the four standard arithmetic operations (+, −,*,/). The TRS-80 uses the same division method we were taught in school, and you remember how long that long division was! So, if we want the TRS-80 to go at top speed, we will have to lessen its work load as much as possible. Amazingly enough, though, an assembly-language routine running this inefficient technique is still faster than even the most efficient BASIC program. When it comes to serious calculation, assembly language is the way to go.

Let's give some thought to improving the division process. We will consider a representative number, 127, and a sample divisor of 11. Upon division the result is 11 with a remainder of 6. As you continue to divide, the next divisor will be 13. 127/13 is 9 with a remainder of 10. Notice that the quotient (9) is now less than the divisor (13), and they had previously been equal. We have passed the point where we need to continue division: The point where the quotient is equal to the square root of the prospective prime, in the case of 127:SQR(127) = 11.2694. Since we need to divide only by odd numbers, this gives us five divisions as compared to 63 using the longer division method.

The Algorithm

So, we arrive at the algorithm: To generate the sequence of prime numbers, we will start at 3 and divide each prospective prime by successive odd numbers until we reach the square root; unless a division comes out even, in which case we will begin testing the next odd number for primeness. If the prospective prime gets through all the divisions up to the square root, it is, in fact, a prime number and is displayed on the screen. The BASIC programs in Program Listings 1 and 2 implement this idea.

Assembly Language Translation

Armed with our BASIC program listings and a copy of *TRS-80 Assembly Language Programming* by William Barden, Jr. (highly recommended), we can proceed to translate our program into assembly language. Several problems stand in our way, among them the necessity of learning a new language, Z-80 (that's the name of the chip doing all the work) assembly language. Assembly language is much more primitive than a high-level language like BASIC, and it takes perseverance to master assembly language. Once you have learned it, though, you will be able to talk to your computer in its own language. Actually, that is not strictly true—the machine's native tongue is just a string of ones and zeros. Each particular combination of 1 and 0 (called bits) performs a specified function.

Some computer wizard decided that it would be much easier for us to understand LD A,(HL) rather than 01111110. Both specify the same operation, namely, to get a byte of data from memory and put it in the computer's accumulator. The first is at least a little more intelligible than the second. A special program called an assembler translates from the first form (assembly language) to the second (machine language). Since assembly language gives you direct control of the machine without the drudgery of memorizing all those bits, it is the language of choice for those who really want to get into the TRS-80 (or any computer, for that matter).

One program you will need when climbing the assembly-language hill is an assembler, preferably with a built-in editor, so that you can edit and then assemble your assembly-language programs. Radio Shack markets EDTASM for that purpose. Microsoft, who developed the Level II BASIC we all know and love, as well as Radio Shack's EDTASM, has come out with an enhanced editor-assembler called EDTASM-PLUS. That is the program I use and recommend to others.

The Z-80 chip is the workhorse of the TRS-80 and is one of the most advanced eight-bit microprocessors in existence. It does not have a division operator, however, and I know of no computer that has a built-in square root. Level II has division and square roots, but like Star Trek, EDTASM, and Blackjack, they are programs; so we will have to program those in, or will we? Since Level II has those functions, can't they be accessed from our assembly-language program? Yes, definitely, but for speed we want our own, since Level II spends a great deal of time on housekeeping and error-detection. Refer to Wes Thielke's article in the February 1980 issue of *80 Microcomputing* for details on interfacing an assembly-language program with Level II BASIC subroutines. It is fortunate that there are many assembly-language programmers who have already written the routines we need. We need only to find them and use them. *80 Microcomputing* readers submit useful routines continually. I found the square root routine in William Barden's "Assembly Line" column for June 1980. The assembly-language program in Program Listing 3 is essentially a translation of the BASIC program in Program Listing 1 with a few changes. First, there is no test for termination of the outer loop; and so the program will run endlessly, cycling through 16 bits. (That is the size of the prime number, NUM, in this program.) To extend this to 24 bits or beyond would require almost a complete reworking of the program.

Users who run this program long enough will notice an apparent error when the number exceeds 32767. The output will then display large negative numbers. What's going on? My program treats all numbers as unsigned positive values, while the output routine in ROM treats the values as signed numbers. In two's complement notation, which is the binary arithemetic system of the Z-80, −32768 follows 32767. To calculate the unsigned value of a negative number in this system, it is necessary to add 65536 to the number. That is why when you use BASIC to calculate Z-80 addresses you will see code similar to: AD = VARPTR(X): IF AD<0 THEN AD = AD + 65536.

Conclusion

Number crunching is not as slow on the TRS-80 as BASIC would have you believe. A properly designed assembly-language routine will show your TRS-80's true colors. The sample program developed here has room for enhancement, which I will leave to those dyed-in-the-wool assembly-language buffs. The output problem mentioned above could be remedied by writing your own output routine. Also, since the square root for the next number will be at most one more than the preceeding square root, the program could be rewritten to account for that rather than computing the square root from scratch each time.

Program Listing 1. *Prime number generator*

Encyclopedia Loader

```
 1Ø :
    '  PRIME NUMBER GENERATOR BY JIM MELLANDER.  11/6/8Ø
 2Ø DEFINT J :
    ' THIS WILL SAME SOME TIME
 3Ø FOR I = 3 TO 9999999 STEP 2
 4Ø  FOR J = 3 TO SQR(I) STEP 2
 5Ø   X = I / J :
      IF X = INT(X)
       THEN
        9Ø
 6Ø   NEXT J
 7Ø  PRINT I, :
     '  NUMBER IS PRIME
 9Ø  NEXT I
1ØØ END
```

Program Listing 2

One-line prime number generator. Note: CINT function is slightly faster than INIT.

```
1Ø :
   '  ONE LINE PRIME NUMBER GENERATOR.  NOTE: CINT FUNCTION IS SLIG
   HTLY FASTER THAN INT.
2Ø DEFINT I,J:
   FOR I = 3 TO 32765 STEP 2:
    FOR J = 3 TO SQR(I) STEP 2:
     IF I / J = CINT(I / J) NEXT I:
     ELSE
      NEXT :
     PRINT I,:
     NEXT
```

Program Listing 3

Assembly-language prime number generator. This program will work only on a Level II machine because the ROM calls are different on Level I. T-BUG users: Enter code at left; punch tape using the command P 4E20 4E9E 4E20 PRIMES.

```
               ØØ1ØØ ;
               ØØ11Ø ;        ASSEMBLY LANGUAGE PRIME NUMBER GENERATOR
               ØØ12Ø ;        BY JIM MELLANDER.
               ØØ13Ø ;
               ØØ14Ø ;        MOST ROM CALLS ARE FROM WES THIELKE'S
               ØØ15Ø ;EXCELLENT ARTICLE IN 8Ø-MICRO., 2-8Ø.  THANKS, WES!
               ØØ16Ø ;
4E2Ø           ØØ17Ø          ORG     2ØØØØ            ;FIT IT ON 4K MACHINE
               ØØ18Ø                                   ;CHANGE ORG TO SUIT
               ØØ19Ø ;
               ØØ2ØØ ;        PRESS ENTER AT MEM SIZE QUESTION.
               ØØ21Ø ;        TO LOAD TYPE: SYSTEM
               ØØ22Ø ;        NAME OF PROGRAM: PRIMES
               ØØ23Ø ;        WHEN LOADED TYPE: /2ØØØØ TO RUN
               ØØ24Ø ;
               ØØ25Ø ;
4E2Ø CDC9Ø1    ØØ26Ø START    CALL    1C9H             ;CLEAR SCREEN
4E23 21Ø3ØØ    ØØ27Ø          LD      HL,3             ;START @ 3
4E26 22874E    ØØ28Ø          LD      (NUM),HL         ;STORE
4E29 1833      ØØ29Ø          JR      PRIME
               ØØ3ØØ ;
               ØØ31Ø ;        MAIN LOOP
```

```
                ;
4E2B 3EØ3       LOOP    LD      A,3             ;TEST VALUE
4E2D 32894E             LD      (TEST),A        ;SAVE IT
4E3Ø 2A874E             LD      HL,(NUM)        ;GET NUMBER
                ;
                ;       SQUARE ROOT ROUTINE FROM 8Ø-MICRO 6/8Ø P. 24
                ;WRITTEN BY JAMES BRAUD.
                ;
4E33 AF                 XOR     A               ;TRY Ø FIRST
4E34 Ø1FFFF             LD      BC,ØFFFFH       ;BC = -1
4E37 Ø9         SQRT1   ADD     HL,BC           ;IF HL < Ø
4E38 3ØØ6               JR      NC,SQRT9        ;       EXIT !
4E3A 3C                 INC     A               ;NEXT ROOT !
4E3B ØB                 DEC     BC              ;UPDATE
4E3C ØB                 DEC     BC              ;       SUBTRACTOR
4E3D 18F8               JR      SQRT1
                ;
4E3F ØØ         SQR     DEFB    Ø
4E4Ø 323F4E     SQRT9   LD      (SQR),A         ;SAVE SQUARE ROOT
4E43 2A874E     DLOOP   LD      HL,(NUM)        ;GET ORIGINAL VALUES
4E46 3A894E             LD      A,(TEST)
4E49 57                 LD      D,A             ;GET DIVISOR IN D-REG
4E4A CD8A4E             CALL    MOD             ;COMPUTE HL.MOD.D
4E4D 7C                 LD      A,H             ;GET REMAINDER IN A-REG
4E4E B7                 OR      A               ;SET FLAGS
4E4F 281E               JR      Z,PRIME1        ;IF Ø REMAINDER NOT PR.
4E51 21894E             LD      HL,TEST         ;POINT TO TEST VALUE
4E54 34                 INC     (HL)            ;ADD 2 TO TEST DIVIDE
4E55 34                 INC     (HL)            ;VALUE
4E56 3A3F4E             LD      A,(SQR)         ;GET SQUARE-ROOT
4E59 BE                 CP      (HL)            ;COMPARE WITH TEST VALUE
4E5A 38Ø2               JR      C,PRIME         ;PRIME IF OVER
4E5C 18E5               JR      DLOOP           ;ELSE LOOP AGAIN
                ;
                ;       NUMBER IS PRIME, SO DISPLAY IT!!
                ;
4E5E 2A874E     PRIME   LD      HL,(NUM)        ;GET PRIME NUMBER
4E61 CD9AØA             CALL    ØA9AH           ;STORE IN ACCUMULATOR
4E64 3E8Ø               LD      A,128           ;PERFORM EDIT
4E66 Ø1ØØØ8             LD      BC,2Ø48D+Ø      ;8*256 8 DIGITS Ø DECIMALS
4E69 CDBEØF             CALL    ØFBEH           ;CHANGE TO DISPLAY FORMAT
4E6C CDA728             CALL    28A7H           ;DISPLAY ON SCREEN
                ;
4E6F 2A874E     PRIME1  LD      HL,(NUM)        ;GET NUMBER TESTED.
4E72 23                 INC     HL              ;ADD 2 TO TRY NEXT NUMBER
4E73 23                 INC     HL
4E74 22874E             LD      (NUM),HL
                ;
4E77 CD2BØØ             CALL    2BH             ;STROBE KEYBOARD
4E7A FEØ1               CP      1               ;BREAK ?
4E7C CA72ØØ             JP      Z,72H           ;BACK TO BASIC
4E7F B7                 OR      A               ;ANY OTHER KEY ?
4E8Ø 28A9               JR      Z,LOOP          ;NO, TRY NEXT NUMBER
4E82 CD49ØØ             CALL    49H             ;WAIT FOR KEYBOARD
4E85 18A4               JR      LOOP            ;JUMP ON KEY PRESS
                ;
4E87 ØØØØ       NUM     DEFW    Ø               ;NUMBER TESTED FOR PRIME
4E89 ØØ         TEST    DEFB    Ø               ;TEST DIVISOR
                ;
                ;       SUBROUTINE TO CALCULATE HL.MOD.D
                ;ADAPTED FROM DIV-16 SUBROUTINE IN 'TRS-8Ø
                ;ASSEMBLY LANGUAGE PROGRAMMING' BY W. BARDEN, JR.
                ;FROM PAGE 197.  CALCULATION OF QUOTIENT REMOVED
                ;
                ;ENTER:         (HL)=DIVIDEND 16 BITS
                ;               (D)=DIVISOR 8 BITS
                ;EXIT           (H)=HL.MOD.D 8 BITS
                ;
4E8A 7D         MOD     LD      A,L
4E8B 6C                 LD      L,H
```

Program continued

```
4E8C 2600      01020         LD      H,0
4E8E 5C        01030         LD      E,H
4E8F 0610      01040         LD      B,16
4E91 29        01050 LOOPM   ADD     HL,HL
4E92 17        01060         RLA
4E93 3001      01070         JR      NC,LOOPM1
4E95 2C        01080         INC     L
4E96 B7        01090 LOOPM1  OR      A
4E97 ED52      01100         SBC     HL,DE
4E99 3001      01110         JR      NC,MOD1
4E9B 19        01120         ADD     HL,DE
4E9C 10F3      01130 MOD1    DJNZ    LOOPM
4E9E C9        01140         RET
               01150 ;
4E20           01160         END     START
```

Program Listing 4

BASIC program to POKE and run the prime number routine.

```
 10 :
    ' BE SURE TO SET MEMORY SIZE TO 19999 BEFORE RUNNING
 20 POKE 16553,255 :
    ' FIX ROM BUG
 30 FOR I = 20000 TO 20126:
     READ X:
     POKE I,X:
     NEXT
 40 POKE 16526,32:
    POKE 16527,78 :
    ' SET UP USR(0) ADDRESS
 50 X = USR(0) :
    ' RUN ROUTINE
100 DATA 205,201,1,33,3,0,34,135,78,24,51,62,3,50,137,78,42
110 DATA 135,78,175,1,255,255,9,48,6,60,11,11,24,248,0,50
120 DATA 63,78,42,135,78,58,137,78,87,205,138,78,124,183,40
130 DATA 30,33,137,78,52,52,58,63,78,190,56,2,24,229,42,135
140 DATA 78,205,154,10,62,128,1,0,8,205,190,15,205,167,40
150 DATA 42,135,78,35,35,34,135,78,205,43,0,254,1,202,114
160 DATA 0,183,40,169,205,73,0,24,164,0,0,0,125,108,38,0,92
170 6,16,41,23,48,1,44,183,237,82,48,1,25,16,243,201
```

TUTORIAL

The Z-80's Hidden Abilities

by Joe Sewell

When Zilog released its extension of the famous Intel 80 series of microprocessors, the micro world was shaken. As is true with similar events, the Z-80 broke barriers for assembly-language programmers. Now the Z-80 is beginning to fall into the same rut as the 8080—its limits are becoming too obvious. The programmer wants more. Unbeknownst to most, there is more to the Z-80 than the documentation tells you.

The Z-80 is of a rare breed in the eight-bit CPU world, because it uses all 256 possible bytes for its instruction set. Even rarer are its many instructions that require two or even three bytes, not including any immediate data. Four bytes, CBH, DDH, EDH, and FDH, are used to begin the multi-byte instructions, and the other byte(s) finishes the code. Not all possible byte combinations are used here, and it is through those unused combinations that the Z-80 programmer can expand his processor's limits. Some of these undocumented codes actually produce a useful response.

I found this information working on a 16K TRS-80 which used the Z-80 processor. I also used Radio Shack's T-BUG monitor with the TSTEP module from Allen Gelder & Co. The TSTEP program, a single-stepper, produces the actual CPU response (unless you change page zero as described in the TSTEP manual), unlike the TRSDOS DEBUG stepper. DEBUG consistently locks up whenever one of these illegal instructions is found, while TSTEP continues as the CPU would. (I verified this by executing a short program consisting of the code under test followed by a return to the monitor.) While I cannot say this will work on all Z-80s, it does work for some.

Codes that use DDH or FDH as the first byte make the most sense. These leading bytes direct instructions that follow which affect the index registers. The instructions following the first byte are actually instructions involving register pair HL. The leading byte directs the instruction to one of the index registers and, with some, adds a displacement byte for indirect addressing. This also works with the undocumented instructions. The basic rules are as follows:

1) If the second byte is one instruction (i.e., not CBH, DDH, EDH, or FDH) and no data follows (such as an LD register, register), the instruction is executed ignoring the leading byte. One exception: If register H is supposed to be used, the most significant byte of the index register is used; if L, then the least significant byte of the index register is used. Thus DD 24 would INC

IX_{HI}, or increment the MSB of register IX. FD 2E would LD IY_{LO},0. (See next rule.)

2) If the second byte is one instruction and would normally have data following (as in a CALL instruction or a LD A,N), the data is assumed to be zeros, and the instruction is executed as such (e.g., a DD 3E would load the accumulator with 00H). So DD 18 would JR 0, which acts like an NOP (except for timing).

3) If the second byte is CB, a four-byte instruction (including the first byte) is assumed. It will act as though the first and third bytes were not there after loading (indirectly) the register to be acted upon using the index register plus a displacement, the third byte being the displacement. FD CB 08 07 would act like LD A,(IX + 8), RLC A (CB 07 is RLC A), saving one byte from the documented method.

4) If the second byte is EDH, the CPU responds with a functional NOP.

As is always true, if DDH is the first byte, register IX is used, and FDH specifies IY. Those undocumented instructions that do not use the index register (i.e., the instructions without the first byte use neither H nor L) will act the same regardless if DDH or FDH is used.

There are few undocumented codes beginning with CBH; they are CB 30–37H. They actually function similar to their CB 20–27H counterparts, but they have one major difference: CB 20–27H are the SLA instructions, which shift each bit in the register left, passing the most significant bit into the carry flag and pushing a zero in the other end. CB 30–37H does the same, except that a one is entered into bit 0 instead of a zero; so they function like an SLA/INC combination.

The illegal EDH instructions are less useful. If byte 2 is less than 40H, greater than BBH, or is between 80H and 9FH inclusive, nothing appears to be done. Those not in the above ranges perform either a NEG, a JP0, or NOP (except for ED 6BH, which does an LD HL,(0)).

These extra operations usually show no other benefit than conserving a byte or two. If the program, for example, requires register pair BC to hold 0000H, you could use the two-byte DD (or FD) 01H instead of the three-byte 01 00 00. Other instructions are very handy. How many times have you wanted to manipulate each byte of the index registers independently of the other byte? Other codes are useless, and the programmer would be better off using the documented code instead of the longer undocumented code.

I would like to emphasize again that these undocumented codes should work on any Z-80 (I only have access to one; so I cannot guarantee other Z-80s will also function this way) when running machine language, but may not work with all monitors and disassemblers. This is probably due to the ways they handle their functions; it can either actually run the present instruction and return to the monitor or read in the instruction and simulate running it. Those that simulate execution probably will not recognize these illegal codes.

UTILITY

KBFIX Your BASIC Programs
File Name
Macros: Let Your Micro Do the Work

KBFIX Your BASIC Programs

by John W. Blattner

Many programmers complain of keybounce problems and the aggravation of loading Radio Shack's KBFIX. I wrote this program as a partial solution to this problem, and I have since found it to be one of the most useful routines in my library. Saver is a machine-language program to be used with BASIC programs. It occupies 244 bytes of low RAM and performs the following two functions:

- Keyboard debouncing.
- Saving of BASIC programs—together with the Saver front end—in SYSTEM format.

You will still have to load Saver before you enter or load a BASIC program. But once you have used the save feature of Saver to record the composite program, you will never have to load Saver again to work with that particular program. Keyboard debounce is permanently installed, without your having to set MEMORY SIZE, much less load a separate program. As an added feature, your BASIC program is now recorded in the more reliable SYSTEM format, which means that you get the two advantages of full six-character titles and a checksum of each 256-byte program block. The checksum feature eliminates all those phony CLOADs—the ones in which you think that you have successfully loaded a 10,000-byte program only to find the BASIC text buffer full of trash.

Make an object tape entitled SAVER with an autostart address of 42EBH. (The correct autostart address is specified by changing line 147 of the listing to read END START.) If you do not own an assembler, you could POKE (using the direct command mode) the 201 required bytes into memory and then use the SAVER program to record itself. POKEing 201 bytes is tedious, but not impossible. Note that all the numbers in the listing are in hexadecimal; these would have to be converted to decimal. Nothing needs to be POKEd into locations 4321H through 434BH. Once the program has been POKEd into memory, type SYSTEM and answer the *? prompt by typing /17131 and pressing ENTER. Your computer should respond with the READY prompt, and you can then record your Saver program by following my directions.

To use Saver, turn on your computer and answer the MEMORY SIZE? query by pressing ENTER. Place the Saver tape in the cassette recorder and rewind it to the beginning. Type SYSTEM and press ENTER. In response to the *? prompt, type SAVER and press ENTER. Depress the play key on the recorder. When the Saver program has been read, another *? prompt will appear. Type / and press ENTER. Your computer will respond with the READY prompt. At this point the keyboard debounce routine has been patched into the computer's operating system, and the SAVE command has been established for later use.

Now enter any BASIC program, either by composing, editing, and debugging it from the keyboard, or by CLOADing it. (Composing will be easier due to the presence of the keyboard debounce routine.) When the BASIC program has been entered and checked, you may record it—along with the Saver front end—in SYSTEM format. To do this, place a blank tape in the recorder and position it for recording. Depress the play and record keys. With the computer in the READY state, type SAVE. The screen will clear, and the query TITLE? will appear. Answer this request with any title of six or fewer characters. (Write this title on the tape that you are recording; it will be needed when you want to load this tape.) Press ENTER, and the recording will commence. When it is finished, you will have a SYSTEM tape with Saver at the beginning, followed by your BASIC program.

To load the new tape, have the computer in the READY state and enter the SYSTEM command. Answer the *? query with the title that you gave the program when you recorded it. When the tape has been loaded, answer the next *? query with / and press ENTER. Your program will then be executed. Keyboard debounce will be built in, and the SAVE command will also be established, in case you wish to modify and rerecord the BASIC program. Once Saver (or any BASIC program with a Saver front end) has been loaded, keyboard debounce and the SAVE command will be there for any new BASIC program. Saver can be used to make copies of itself. Simply follow the directions for saving a BASIC program, with no BASIC program segment.

When you first load a composite (Saver-BASIC) program, you *must* run it from the SYSTEM state. After it has been run once, however, you may use either the SYSTEM command (with a transfer to address 17131) or the RUN command for subsequent execution. You may also edit the BASIC part of the program in the usual fashion.

The 43 (decimal) bytes of empty space at line 34 of the Program Listing provide stack space for the ROM initialization procedure. This feature can be useful in case of a program crash—most likely caused by other assembly-language routines that you are using with your BASIC program—resulting in ROM reinitialization. After such a crash, when your computer returns to the READY state, enter the SYSTEM command and transfer to 17131. With a bit of luck, your BASIC program will be restored and begin to execute. If you have one of the late model ROMs that has KBFIX built in, you may still want to use Saver for its other features. It will function properly with your ROM; its keyboard debouncer will substitute for that of the ROM.

Program Listing. *Saver*

```
42EB          00000         ORG     42EBH
              00010 ;         ***SAVER PROGRAM***
              00020 ;FIRST SEGMENT ESTABLISHES LINKAGES FOR
              00030 ;KBFIX AND SAVE ROUTINES, SETS POINTERS
              00040 ;FOR BASIC PROGRAM, AND RUNS THE LATTER.
              00050 ;
42EB 210F43   00060 START   LD      HL,KBENT
42EE 221640   00070         LD      (4016H),HL
42F1 216A43   00080         LD      HL,TAPE
42F4 22A141   00090         LD      (41A1H),HL
42F7 21DD43   00100         LD      HL,FINIS
42FA 22A440   00110         LD      (40A4H),HL
42FD 11FAFF   00120         LD      DE,0FFFAH
4300 CD2C1B   00130         CALL    1B2CH
4303 23       00140         INC     HL
4304 23       00150         INC     HL
4305 22F940   00160         LD      (40F9H),HL
4308 211E1D   00170         LD      HL,1D1EH
430B E5       00180         PUSH    HL
430C C35D1B   00190         JP      1B5DH
              00200 ;
              00210 ;
              00220 ;THE NEXT SEGMENT IS A VERSION OF RADIO
              00230 ;SHACK'S KBFIX PROGRAM.  THE GAP IN THE
              00240 ;MIDDLE PROVIDES STACK SPACE FOR THE ROM
              00250 ;DURING INITIALIZATION.
              00260 ;
430F 010138   00270 KBENT   LD      BC,3801H
4312 1600     00280         LD      D,0
4314 213640   00290         LD      HL,4036H
              00300 ;
4317 1838     00310         JR      NXB
4319 54       00320 TITLE   DEFM    'TITLE? '
4320 00       00330         DEFB    0
002B          00340         DEFS    2BH
              00350 ;
434C 2C       00360 TSM     INC     L
434D 14       00370         INC     D
434E CB01     00380         RLC     C
4350 F8       00390         RET     M
4351 0A       00400 NXB     LD      A,(BC)
4352 5F       00410         LD      E,A
4353 AE       00420         XOR     (HL)
4354 73       00430         LD      (HL),E
4355 A3       00440         AND     E
4356 28F4     00450         JR      Z,TSM
4358 5F       00460         LD      E,A
4359 C5       00470         PUSH    BC
435A 018005   00480         LD      BC,580H
435D CD6000   00490         CALL    60H
4360 C1       00500         POP     BC
4361 0A       00510         LD      A,(BC)
4362 A3       00520         AND     E
4363 C8       00530         RET     Z
4364 7A       00540         LD      A,D
4365 07       00550         RLCA
4366 07       00560         RLCA
4367 C3FE03   00570         JP      3FEH
              00580 ;
              00590 ;
              00600 ;THE LAST SEGMENT DUMPS MEMORY FROM 42EB
              00610 ;TO THE END OF THE BASIC PROGRAM IN SYS-
              00620 ;TEM FORMAT.  IT IS CALLED BY "SAVE".
              00630 ;
436A CDC901   00640 TAPE    CALL    1C9H
              00650 ;PRINT "TITLE? " ON SCREEN.
436D 211943   00660         LD      HL,TITLE
```

Program continued

```
4370 CDA728              CALL    28A7H
                   ;FILL INPUT BUFFER WITH SPACES.
4373 21E841              LD      HL,41E8H
4376 E5                  PUSH    HL
4377 0606                LD      B,6
4379 3620          FILL  LD      (HL),20H
437B 23                  INC     HL
437C 10FB                DJNZ    FILL
437E E1                  POP     HL
                   ;INPUT THE TITLE.
437F 0606                LD      B,6
4381 CDD905              CALL    5D9H
                   ;TURN ON CASSETE AND WRITE LEADER.
4384 AF                  XOR     A
4385 CD1202              CALL    212H
4388 CD8702              CALL    287H
                   ;ESTABLISH RST 8 FOR CASSETTE OUTPUT.
438B 216402              LD      HL,264H
438E 220140              LD      (4001H),HL
                   ;WRITE TAPE TYPE IDENTIFIER.
4391 3E55                LD      A,55H
4393 CF                  RST     8
                   ;RECORD TITLE.
4394 21E841              LD      HL,41E8H
4397 0606                LD      B,6
4399 7E            NAME  LD      A,(HL)
439A FE0D                CP      0DH
439C 2002                JR      NZ,SKIP
439E 3E20                LD      A,20H
43A0 CF            SKIP  RST     8
43A1 23                  INC     HL
43A2 10F5                DJNZ    NAME
                   ;COMMENCE RECORDING PROGRAM.
43A4 21EB42              LD      HL,START
43A7 E5                  PUSH    HL
                   ;RECORD ONE BLOCK OF 256 BYTES.
43A8 1E00          LP1   LD      E,0
43AA 4B                  LD      C,E
43AB 3E3C                LD      A,3CH
43AD CF                  RST     8
43AE 7B                  LD      A,E
43AF CF                  RST     8
43B0 7D                  LD      A,L
43B1 CF                  RST     8
43B2 7C                  LD      A,H
43B3 CF                  RST     8
43B4 85                  ADD     A,L
43B5 83                  ADD     A,E
43B6 4F                  LD      C,A
43B7 7E            LP2   LD      A,(HL)
43B8 CF                  RST     8
43B9 81                  ADD     A,C
43BA 4F                  LD      C,A
43BB 23                  INC     HL
43BC 1D                  DEC     E
43BD 20F8                JR      NZ,LP2
43BF 79                  LD      A,C
43C0 CF                  RST     8
                   ;CHECK FOR END.
43C1 ED5BF940            LD      DE,(40F9H)
43C5 DF                  RST     18H
43C6 38E0                JR      C,LP1
                   ;RECORD END SYNC BYTE.
43C8 3E78                LD      A,78H
43CA CF                  RST     8
43CB E1                  POP     HL
                   ;RECORD AUTOSTART ADDRESS.
43CC 7D                  LD      A,L
43CD CF                  RST     8
43CE 7C                  LD      A,H
```

```
43CF CF        01370          RST     8
               01380 ;TURN OFF CASSETTE.
43D0 CDF801    01390          CALL    1F8H
               01400 ;RESTORE NORMAL RST 8.
43D3 21961C    01410          LD      HL,1C96H
43D6 220140    01420          LD      (4001H),HL
               01430 ;RETURN TO BASIC.
43D9 C3191A    01440          JP      1A19H
43DC 00        01450          DEFB    0
43DD 0000      01460 FINIS    DEFW    0
0000           01470          END
00000 TOTAL ERRORS
```

File Name

by Theodore J. LeSarge

Have you ever gone back to work on a program and discovered that you forgot the file name? File Name is a program I have developed to remedy this problem.

I have developed two versions of this program. The first is for machine-language programmers, and the second is listed in BASIC. The machine-language version is somewhat better because it is relocatable and contains brief instructions. It loads well below stack space so you don't need to set the memory size. The BASIC version includes no prompts and requires that you rerun the BASIC program for each use. I designed both versions for the TRS-80 Model I Level II with 16K. Because the program is ROM dependent, I don't know if it will run in the Model III.

In Program Listing 1, you can see that the first few lines of the machine-language program clear the screen, set up the cursor, load the HL register pair with the starting address of the first message, and display it. Location 28A7H in ROM requires that the HL pair be loaded with this starting address. Note that the messages end with a double quotation mark, 22H, the delimiter to stop the ROM display routine. A DEFB 0 will also halt output. The next three lines are a keyboard scan to hold the CPU until you are ready for further operation. Execute the keyboard scan by pressing ENTER.

Lines 200 through 300 are the heart of the program. First they clear the screen and set the A register to zero so that the call to 0212H will select the first cassette. Line 320 calls ROM to turn on the cassette and checks for the synchronization byte to appear. Since file names can be a maximum of only six letters, line 330 sets up a loop to read seven bytes from the tape. It displays seven bytes, of which only the last six make up the actual file name. The first letter is the file name header code which is 55H or the letter U. The program removes this letter from the display by writing over it with the second message.

Line 340 assigns the HL pair to a screen location. The call to 0235H reads a byte from tape and stores it in the A register. The following three lines store the byte in video memory, bump the HL register, and decrement the count in the B register. The program checks for a zero, and if the zero flag is set, calls 01F8H to turn off the cassette.

Lines 410 through 520 display the remaining three messages. The message in line 620 has two spaces at the end. The second space removes the U that I mentioned earlier. The last few lines go into another keyboard scan and wait for you to press ENTER or the letter E. ENTER sends the program back to the start, and pressing E returns to ready.

To make the BASIC listing short, I eliminated all the messages from the display. This version, shown in Program Listing 2, POKEs the proper machine code into memory, adds the data, and checks at the end to see if everything was read correctly. It then sets the location for the USR function to jump to. After it displays the file name, it loads the A register with a space and stores it on the screen to remove the file name header code (the U again). The program then returns to ready.

One important thing to remember is that, if your cassette player is a CTR-80, a pop may appear on the tape when the cassette motor is shut off. A Radio Shack modification eliminated this problem. If you haven't had your cassette player modified, by all means do so. You can still play it safe and remove the remote plug from your cassette. This will keep the machine from turning off and possibly ruining a tape when you use this program.

Encyclopedia Loader™

Program Listing 1. *File Name, machine-language version*

```
                 ;********************************
                 ;*** FILE NAME ***
                 ;BY THEODORE J. LESARGE
                 ;ORIGINAL PROGRAM - 1/20/80
                 ;REVISED - 4/11/81
                 ;********************************


7F58                     ORG     7F58H           ;32600
7F58 CDC901              CALL    01C9H           ;CLS
7F5B 21003D              LD      HL,3D00H        ;LINE FOUR
7F5E 222040              LD      (4020H),HL      ;SET UP CURSOR
7F61 21B97F              LD      HL,$+58H        ;SET UP MESSAGE 1
7F64 CDA728              CALL    28A7H           ;  & DISPLAY IT

7F67 CD4900              CALL    049H            ;KBD SCN
7F6A FE0D                CP      0DH             ;ENTER?
7F6C 20F9                JR      NZ,$-05H        ;NO? AGAIN

7F6E CDC901              CALL    01C9H           ;CLS
7F71 AF                  XOR     A               ;ZERO A REGISTER
7F72 CD1202              CALL    0212H           ; FOR 1ST CASSETTE
7F75 CD9302              CALL    0293H           ;ON TAPE/FIND SYNC BYTE
7F78 0607                LD      B,07H           ;READ 7 BYTES
7F7A 214E3D              LD      HL,3D4EH        ;SCREEN LOCATION
7F7D CD3502              CALL    0235H           ;READ A BYTE
7F80 77                  LD      (HL),A          ;STORE ON SCREEN
7F81 23                  INC     HL              ;BUMP DISPLAY
7F82 10F9                DJNZ    $-05H           ;B=0? NO-GO AGAIN
7F84 CDF801              CALL    01F8H           ;OFF CASSETTE

7F87 21403D              LD      HL,3D40H        ;LINE 5
7F8A 222040              LD      (4020H),HL      ;SET CURSOR
7F8D 21C87F              LD      HL,$+3BH        ;2ND MESSAGE
7F90 CDA728              CALL    28A7H           ;DISPLAY IT
7F93 21C03D              LD      HL,3DC0H        ;LINE 7
7F96 222040              LD      (4020H),HL      ;SET CURSOR
7F99 21D87F              LD      HL,$+3FH        ;MESSAGE 3
7F9C CDA728              CALL    28A7H           ;DISPLAY IT
7F9F 21003E              LD      HL,3E00H        ;LINE 8
7FA2 222040              LD      (4020H),HL      ;CURSOR
7FA5 21EF7F              LD      HL,$+4AH        ;MESSAGE 3 PART 2
7FA8 CDA728              CALL    28A7H           ;DISPLAY IT

7FAB CD4900              CALL    049H            ;KBD SCN
7FAE FE0D                CP      0DH             ;ENTER?
7FB0 28A6                JR      Z,$-58H         ;GO WAAAAY BACK
7FB2 FE45                CP      45H             ;"E"?
7FB4 CACC06              JP      Z,06CCH         ;READY IN ROM
7FB7 18F2                JR      $-0CH           ;GO AGAIN

7FB9 52                  DEFM    'READY CASSETTE"
7FC8 46                  DEFM    'FILE NAME IS:  "
7FD8 41                  DEFM    'AGAIN? - PRESS <ENTER>"
7FEF 45                  DEFM    'END? - PRESS <E>"
7F58                     END     7F58H
00000 TOTAL ERRORS
```

Program Listing 2. *File Name, BASIC version*

```
10 C = 0:
   B = 0
```

```
 20 FOR A = 30000 TO 30030
 30  READ B:
     POKE A,B
 40  C = B + C:
     NEXT
 50 IF C < > 2963
     THEN
      PRINT "BAD LOAD!":
      END
 60 DATA 205,201,1,175,205,18,2,205,147
 70 DATA 2,6,7,33,78,61,205,53,2,119,35
 80 DATA 16,249,205,248,1,62,32,50,78
 90 DATA 61,201
100 POKE 16526,48:
    POKE 16527,117
110 CLS :
    X = USR(0):
    END
```

Macros: Let Your Micro Do the Work

by John R. Hind

Macro programming is nearly as old as the digital computer itself. In the days when assembly language was the wave of the future for the programmer, someone noticed that a lot of programs had sections of code that were almost identical and came up with the idea of duplicating card decks (or was it paper tape?) with these sequences so that they could drop them into a new program.

In time, these sequences were transcribed on tape and pulled in by a utility routine where needed in a source file. The immediate problem was that the one or two odd cards in an imbedded sequence could no longer be changed by selective duplication on a key punch, so the concept of substitution was born. At first the utility control statement which brought in the sequence was preceded by several EQUate statements for the changeable symbols in the sequence. Next, the utility control statement was expanded to include definitions of symbols used in the sequence. Finally the concept of character string replacement came into the picture. The source line which brought in the sequence was termed a macro-instruction. It named a skeletal definition to be inserted that defined one or more parameters which the supplied value would replace before insertion.

Since some instruction sequences were similar but not exactly alike, the concept of conditional inclusion and looping was added to the macro definitions. With this new flexibility came the need to parse the macro parameters and to pass information from one macro to another. The concept of the macro-time variable known as the set-symbol and statements which could compute its value as either a character string or as a character number finally satisfied this need. With this new form, macros became the mainstay of the operating system, not only as a means of simplifying user interface to services like I/O (OPEN, GET, PUT, and the like), but also for generation and configuration of the system itself.

Macro concepts have made their way into almost every computer language system in existence—from the preprocessor that created structured FORTRAN through PL/1 to most of the command languages that time-sharing systems use. The reason for this wide use is that the concept simplifies the human interface and saves user time and effort.

When I found myself in the middle of another project, typing in repetitive lines of assembler source code, I finally decided that it was time to let my micro do some of the tedious work. I needed a macro processor, but it had to do more than process assembler source. I wanted something that I could use on text, data, and BASIC. It had to have the ability to allow a macro call in the middle of a line, and to provide full character string manipulation and a reasonable level of arithmetic functions. I didn't care how many cycles of my Z-80 it chewed up, what was important to me was the number of instructions executed during that time.

I proceeded to define my personal macro language, trying to balance the function I was looking for against the complexity of implementing it in BASIC. I came up with a reasonable set of language constructs (see the Micro-Mac Summary section in this chapter). The program in the Micro-Mac Source section is an implementation of my macro language. It is not an end, but rather a starting point.

Many large machines have instruction sets which are orders of magnitude more powerful than the Z-80 and hence much easier to program. A typical large processor usually has some form of a storage-to-storage MOVE instruction, so let's define our own macro version:

```
.REM move characters from addr P2 to addr P1 for a
.REM length of P3 --ex-- .MOVE  A,B, 55
.MAC MOVE
  PUSH HL ;&0 &1,&2,&3
  PUSH DE
  PUSH BC
.CIF= DEST-IN-DE,&1,DE
  LD DE,&1
.: DEST-IN-DE
.CIF= SOURCE-IN-HL,&2,HL
  LD HL,&2
.: SOURCE-IN-HL
.CIF= COUNT-IN-BC,&3,BC
  LD BC,&3
.: COUNT-IN-BC
  LDIR
  POP BC
  POP DE
  POP HL
.MEND
```

If the above macro was in a file named MACLIB, and we typed the following lines after running the BASIC program discussed in the Micro-Mac Source section:

```
.IM MACLIB
.OUTF TESTMV
; NOW LETS TEST THE MOVE MACRO
.MOVE TOAD,FROMAD,80
; END OF THE TEST
.OUTF *DI
.MEND
```

Then we would find the following lines in a file named TESTMV:

```
; NOW LETS TEST THE MOVE MACRO
  PUSH HL ;.MOVE TOAD,FROMAD,80
  PUSH DE
  PUSH BC
  LD DE,TOAD
```

```
                        LD HL,FROMAD
                        LD BC,80
                        LDIR
                        POP BC
                        POP DE
                        POP HL
                      ; END OF THE TEST
```

You may have noticed that if DE, HL, or BC had been the to or from or length parameters, then the output would not have included the respective load of the DE, HL, or BC register. This is a straightforward case of conditional branch logic in a macro. Since the GOTO, IF, and LABEL commands are executed in macros as a result of substitution, it is possible to define conditional logic which is self-modifying. The simplest example of this can be seen in a structured programming class statement such as the GOTO command in PASCAL:

```
.REM TRDOS DISK I/O CALLS,
.REM     P1 IS DCB ADDR
.REM     P2 IS ADDR OF BUF/REC OR REC NUMBER
.REM     P3 IS OPEN RECORD LENGTH
.MAC INIT,OPEN,POSN,READ,WRIT,CLOS,KILL,VERF
   LD DE,&1 ; LOAD DE WITH THE DCB ADDR
.GOTO &0
.: .POSN
   LD BC,&2 ; LOAD BC WITH RECORD NUMBER
   CALL 4442H ; &0
.MRET
.: &0
.CIF= &0,+&2,+
   LD HL,&2 ; LOAD HL WITH ADDR OF BUFFER/RECORD
.GOTO &0
.: .INIT
   CALL 4420H  ; &0
.MRET
.: .OPEN
   LD B,&3 ; RECORD LENGTH
   CALL 4424H  ; &0
.MRET
.: .READ
   CALL 4436H  ; &0
.MRET
.: .WRIT
   CALL 4439H  ; &0
.MRET
.: .CLOS
   CALL 4428H  ; &0
.MRET
.: .KILL
   CALL 442CH  ; &0
.MRET
.: &0
; **** BAD CALL *****
.MEND
```

As you can see, I used a single skeletal definition and gave it a set of different macro names. When this macro is called, its name is placed in P0 (substitution symbol &0), which is later used as a label of a GOTO to simulate a CASE statement. Then you give each clause the appropriate label value corresponding to the respective P0 value. If no case clause with a matching label is found, a label with the P0 value will stop the search by marking the end of the case statement.

The TRSDOS disk I/O macro can save you keystrokes as well as decrease the time you spend searching in the reference manual for those all important hex addresses. Let's add a macro which defines a DCB area and optionally places a file name in it:

```
.REM this macro defines a DCB for a TRDOS file
.REM P1 is the label of used for the area
.REM P2 is the optional file name
.MAC DCB
.GOTO +&2
.: +
&1 DEFB 03H
   DEFS 31
.MRET
.: +&2
.LEN 9,'&2'
.ASET 9,31,-,&9
&1  DEFM  '&2'
.AIF< FINISH,&9,0
    DEFB 03H
.AIF= FINISH,&9,0
    DEFS  &9
.: FINISH
.MEND
```

The important part of this macro is the use of .LEN, to set P9 to the length of P2, and the .ASET which computes the remaining length to be defined in the DCB after the file name has been placed there (31-P9). Notice that the macro used parameter P9 as the work variable so that no global variables would be touched.

We could have done this in fewer macro statements by remembering that the file name doesn't have to end in 03H if the rest of the DCB is blank. You might ask how that helps:

```
.MAC DCB
.SETC 9,'&2                                ',1,32
&1 DEFM '&9'
.MEND
```

Notice we have made use of the .SETC statement which provides us with a substring function. By concatenating P2 with 32 blanks and then using the first 32 characters of the expression, we have computed the proper value for the DEFM expression. There is a lesson in this: Never underestimate the power of a pure character string expression, which is what macros are all about.

There is one other important concept that we need to illustrate: the use of a second output file. Many times it is useful to define table entries at the point in the code at which we first use them, but actually place them all at the end of a program. Micro-Mac will allow switching back and forth between a primary and a secondary output file. At the end of the code, where the table should be inserted, we can close the secondary file and imbed it in the primary file. The following macro can assist this process:

```
.MAC STAB PUTT GETT
.GOTO &0
.: STAB
```

```
.OUT2 TABLE
.OUTF
.MRET
.: PUTT
.OUT2
&1
.OUTF
.MRET
.: GETT
.OUT2 *DI
.OUTF
.IM TABLE
.MEND
```

The STAB call will open a secondary output file named Table. The PUTT call will place P1 in the Table file as a single line each time it is called. The GETT call will close the secondary file and imbed it at the point of call. Let's look at a part of a sample program which uses these calls:

```
.STAB (OPEN TABLE FILE)
START  LD (PA),HL
.PUTT PA DEFW 0 ; THE ZZZZZ PARAMETER
       LD (FLAG),A
.PUTT FLAG DEFB 0 ; INIT FLAG - 0=> XXX - 1=> YYY
  ....
       RET ;FINISHED
; DEFINE DATA AREAS
.GETT (INSERT PA AND FLAG DEFINITIONS)
       END START
```

So far I have restricted my examples to assembly-language usage. You can use Micro-Mac on other forms of text data as well. Let's look at a text macro which writes a form letter. (It uses a : symbol to display the paragraph symbol and a { for a page symbol.)

```
.REM P1 is a name ,P2 is street,P3 is city,
.REM P4 is state, P5 is ZIP, P6 is interest
.MAC FORM
.INDX 9,&1,' '
.SETC 9,&1,1,&9
§-
TRS-80 Computer Club
Z80 Micro Lane
Somewhere, USA   9999

&1
&2
&3, &4   &5

Dear &9:
     We are happy to announce that &6 will be one of -
the subjects of presentations at our next meeting -
on &A night &B.  The meeting will start at &C in -
&D which is located at &E.

Sincerely;
402DH
.MEND
```

In this case the input file would use .SETC symbols to define global variables A, B, C, D, and E for all form letters and then would include a .FORM call for each person who is to receive the letter:

```
.SETC A,friday
.SETC B,'January 23rd, 1981 '
.SETC C,7:30 pm
```

```
.SETC  D,John Brown's home
.SETC  E,1492 Discovery lane
.FORM  Tom Jones,862 Wild Rd.,Xvile,N.Y.,99999,graphics
.FORM  ....
.FORM  ....
```

You can see that the technique involved in macros has a wide range of applications outside of a macro-language processor such as Micro-Mac. The general misconception is that use of this technique must be limited to a large computer system.

Micro-Mac Summary

The following are descriptions of commands that can be used with Micro-Mac.

Input Line Convention

An input line is defined as a string of characters which ends with an ENTER key (0DH). If the line is taken from an EDTASM format file, the leading line number will be removed before the line is processed. The trailing ENTER is removed from the line if the string ends in a dash (—) as is the dash itself. If the string ends in a semicolon (;), this character will be removed. If a dash is desired at the end of a string in a line ending with an ENTER key, then the string should end with —;. If the line does not end with an ENTER key, the string should end with ——. Similarly, if a ; is desired at the end, the line should be terminated by ;;.

Each input line is separately processed, and if it does not contain a macro command, it will be written to the output file. Since records in the output file are delimited by ENTER characters, several input records can be concatenated into a single output record by use of the dash convention.

A macro invocation is signaled by a period, or dot (.) in the first position of the input line string. If the line begins with two dots (..), then one is removed, and the line will not be scanned as a macro command.

Substitution

Each input record (line string and optional ENTER) is first processed by a symbolic substitution algorithm. The line is searched for an ampersand character (&), and if one is found, the character following it is treated as a Micro-Mac variable name, and both of these characters are replaced by the current value string of that variable. The total length of the resulting string including the optional ENTER key must be less than 255 characters. This means that care must be used with respect to the size and number of substitutions made in a record.

If two ampersand characters are found (&&), a single & will be left in the record but will not be used as a trigger for substitution. If the variable A had the value CATS, the string "JOHN && MARY LOVE &A" would become "JOHN & MARY LOVE CATS" by the substitution process.

Variables

A variable is defined as a single uppercase character and is denoted by {var} in the following descriptions. There are ten local variables denoted by the digits 0, 1, . . . , 9 and 32 global variables denoted by the values :, <, = , >, ?, A, B, C, . . . , Z. A variable may contain a character-string value having a length of up to 246 bytes. Micro-Mac provides built-in macros that can set variable values based on parameter strings resulting from substitution of input records.

Macro Format

A macro invocation is defined as an input record which, after substitution, contains a dot (.) as its first character and is terminated by ENTER. The name of the macro follows the dot and is separated by at least one space from the parameters. Here is the general format for a macro invocation:

```
.{macroname} <{p1}<,{p2}<,...<,{p9}>>>>ENTER
```

The parameter list consists of up to nine strings separated by commas (,). If the string contains leading spaces or commas, it must be enclosed in single or double quotation marks:

```
.TEST XYZ," marry's hat ",'(HL),I',A B C ENTER
```

A macro is a set of named input lines that can be imbedded in the input stream including an invocation line at that point. Before inclusion is made, the local variables 0, . . . , 9 are placed on a stack and then are set to equal the value of the macro name string and any parameters which appeared on the invocation line. In the above example, the local variables would be set as follows:

```
"0" => {.test}
"1" => { marry's hat }
"2" => {(HL),I}
"3" => {A B C}
```

When the logical end of the named lines is found, the local variables will be reset to their original values before scanning of the input stream continues.

Macro Definition

Two built-in macros are provided to allow macros to be defined from the input stream. .MAC signals the start of a macro definition and gives the macro from one to nine names. All lines following .MAC up to .MEND are placed without substitution into an internal table for later use. Here is the general format:

```
.MAC {name}<,{name}<,....>>ENTER
   {macro text lines}
.MEND ENTER
```

Let's define a macro, for example, which would add the contents of the storage location named in P2 to the storage location named in P1:

```
 .MAC ADDS
    LD   DE,(&2)
    LD   HL,(&1)
    ADD  HL,DE
    LD   (&1),HL
.MEND
```

Built-in Macros to Set Variable Values

SETC: Sets {var} to MID$({string value},{start},{length}). Default is the first character in the string and the string's length. An example of the format follows:

```
.SETC {var},{string value}<,{start}<,{length}>>ENTER
```

LEN: Sets {var} to the length of {string value}. Here is the format to use:

```
.LEN {var},{string value}ENTER
```

INDX: Sets {var} to INSTR ({string value},{object}). The format for INDX commands is:

```
.INDX {var},{string value},{object}ENTER
```

SETA: Sets {var} to the arithmetic combination of {num} as computed in a left to right sequence using {opr} where {opr} can be +, −, *, /, OR, or AND (i.e., [.SETA Z,&A, + ,35,*,&B] means Z = ((A + 35)*B). Here is the general format:

```
.SETA {var},{num}<,OPR,{num}<,...>>
```

SEQN: Assigns a number to a variable that is incremented by 1 each time this macro is called. The format for SEQN commands is as follows:

```
.SEQN {var}ENTER
```

PSET: Resets local variables 1, . . . , 9 to {new 1}, . . . ,{new 9}. Here is the format to use with PSET:

```
.PSET {new 1}<,{new 2}<,{new 3}<, . . . >>>ENTER
```

INPT: Prompts the operator for an input line and assigns the response to {var}. The general format is:

```
.INPT {var},{operator message}ENTER
```

Sequence Control—Built-in Macros

Colon: Used to identify a position in the input source. The {label} can be a result of substitution, hence positions can be dynamically computed. Here is the general format:

```
.: {label}ENTER
```

GOTO: Causes a search for the corresponding line containing a [.: {label}] within the current input file or macro. When the line is found, processing continues. If issued in a file context, GOTO has the effect of a search until the line is found or the end of the file is reached. The format is:

```
.GOTO {label}ENTER
```

CIF: String {value 1} and string {value 2} are compared, and if the result is {?}, the program will GOTO {label}. The {?} can be =, <, or >. The format for a CIF command is:

```
.CIF{?} {label},{value 1},{value 2}ENTER
```

AIF: Same logic as CIF except the arithmetic values of string {value 1} and string {value 2} are compared. Here is the format:

```
.AIF{?} {label},{value 1},{value 2}ENTER
```

Input File Definition

IM: Will imbed the named TRSDOS file {file name} or request input from the operator if {file name} is *KI. The current input source remains open and will be used for input when the end of the imbedded file is found. If 15 files have been reserved for BASIC, then up to 13 imbeds of TRSDOS files may be nested. The format for an IM command is as follows:

```
.IM {file name}ENTER
```

The files may be either in ASCII format or NEWDOS EDTASM format. When using the latter format, the line number will be removed automatically. If the line starts with a tab, it will be replaced by two spaces.

Output Files

OUTF: Will cause macro output to be directed to the primary output file. If a file name is given, any previous primary output file will be closed, and the named file will be opened. If the EDTASM option is specified, the {filename} is created in NEWDOS EDTASM format. When this option is used, the first six characters of {filename} must not contain special characters (i.e., TEST and TESTOK/ASM.pass:1 are good names, but TEST/ASM will cause errors because of the / in position 5). Here is the general format for the OUTF command:

```
.OUTF <{filename}<,EDTASM>>ENTER
```

OUT2: Same as .OUTF but for secondary output. Note that the EDTASM option is *not allowed*. Here is the format for this command:

```
.OUT2 <{filename}>ENTER
```

Options

OPTN: Sets the option flag. A flag of 0 is normal while a flag of 1 causes a trace of input lines obtained from files or macros, and a trace of output strings. The format is:

.OPTN {option number}

Micro-Mac Source

This version of Micro-Mac runs on a 48K TRS-80 single disk system. About 5K bytes of unused space are available for the addition of new macro processor features, and about 18K can be used as a macro definition line buffer. It can be run on a 32K system by reducing the size of the string area and giving up some of the patch space. For a listing of the BASIC variables, see Table 1 at the end of the article.

```
100 CLEAR 20000 : DEFINT I-N : DEFINT Z,S : DEFSTR A-H :
      DEFSTR P
```

Line 150 sends the program to the subroutine beginning at line 4800 which initializes the variables—this code is near the end of the program to reduce GOTO and GOSUB search time.

```
150 GOSUB 4800
200 CLS : PRINT"MICRO-MACRO PROCESSOR  VERSION 1.0"
250      PRINT"          BY JOHN R. HIND"
300 PRINT CR;CR;CR;"ENTER MACRO LANGUAGE SOURCE"
350 PRINT "(.OUTF FN /.IM FN /ETC..)
```

The program then branches to the main execution loop with the keyboard and display set up as the I/O devices. This way, the operator can enter macro calls as program controls which avoids providing extra code in Micro-Mac for this function and describing yet another user interface.

```
400 GOTO 1600
```

This parse subroutine will extract a macro name and its parameters from the current line buffer BL and place these string values on top of the parameter stack P using the stack index IP. The first line will strip out the macro name.

```
500 J=INSTR(BL,BK) :
      IF J>Z0 THEN
      P(IP)=LEFT$(MID$(BL,Z1,J-Z1)+"      ",Z5) :
      BL=MID$(BL,J+Z1,LEN(BL)-J-Z1) :
      ELSE P(IP)=LEFT$(LEFT$(BL,LEN(BL)-Z1)+"      ",Z5) : BL=""
```

Now we will clear space for the macro parameters in case fewer than nine parameters appear on the line and will initialize the loop counter J which will be used to index the parameter stack.

```
550 FOR L=Z1 TO Z9 :
      P(IP+L)="" :
      NEXTL :
      J=Z0
```

We will loop until the line buffer is empty, saving all parameters that we might find. Note that all leading blanks are ignored.

```
600 L=LEN(BL) :
     IF L=Z0 THEN RETURN :
     ELSE A=LEFT$(BL,Z1) : BL=MID$(BL,Z2) :
      IF A=BK THEN 600
```

Once we have found a parameter, we must check to see if it is null or if it is delimited by a special character (’ or :).

```
650 J=J+Z1 :
     IF A="," THEN 600 :
      ELSE IF A<>"´" AND A<>" : " THEN 750
```

A normal string is a string which is between commas. Put it in parameter stack P.

```
700 K=INSTR(BL,A) : P(IP+J)=LEFT$(BL,K-Z1) :
     BL=MID$(BL,K+Z2) :
      GOTO 600
```

A special string is a string which contains delimiter characters. Remove delimiter characters.

```
750 K=INSTR(BL,",") :
     IF K=Z0 THEN K=L
800 P(IP+J)=A+LEFT$(BL,K-Z1) : BL=MID$(BL,K+Z1) : GOTO 600
```

This substitution subroutine replaces all macro variables found in the line buffer, BL, with their current string values. We will print the unedited line if the TRACE option flag is on.

```
900 J=Z1 :
     IF (JT AND Z1)=Z1 THEN
      IF S(IS)<>Z0 THEN
      PRINT " : : ";S(IS);" : : ";BL;" : : "
950 J=INSTR(J,BL,BA) :
     IF J=Z0 THEN RETURN :
      ELSE K=ASC(MID$(BL,J+Z1))
```

If the variable is &, we want to leave an & in the line buffer and not rescan it. That is, we must not use it as a trigger to find a variable name.

```
1000 IF K=ZA THEN
      BL=LEFT$(BL,J)+MID$(BL,J+Z2) : J=J+1 : GOTO 950
```

Normal variables are saved on the very bottom of the parameter stack P, while the current parameters 0 through 9 are on top of the stack at index IP. Variable K is set to the index in P which contains the string value.

```
1050 IF K<58 THEN K=IP+K-48 :
      ELSE IF K>95 THEN K=K-90 :
        ELSE K=K-58
```

Replace the trigger, &, and the variable name with the string value.

```
1100 BL=LEFT$(BL,J-Z1)+P(K)+MID$(BL,J+Z2)  :  GOTO 950
```

The GET-NEXT-LINE subroutine gets the next line from the current input source, defined by index IS, in the source stack S. S(IS) will be either a positive value which indicates that the next line is from the macro buffer G, zero for keyboard, or negative for a file.

```
1200 J=S(IS)  :
     IF J>Z0 THEN BL=G(J)  :  S(IS)=J+Z1  :  RETURN
```

Line 1200 gets a line from keyboard or file. If the line received is the end of file, then return a ".MEND" which causes termination of the source statements in the stack.

```
1250 J=J AND &HF :
     IF J=Z0 THEN LINEINPUT"SOURCE INPUT : ";BL :
      ELSE IF EOF(J) THEN BL=".MEND" :
        ELSE LINEINPUT#J,BL : GOSUB 1400
```

At the end of line 1250 we will handle flags and insert an ENTER character if necessary.

```
1300 A=RIGHT$(BL,Z1)  :
      IF A=BS THEN A=CR :
      ELSE IF A=BD THEN A=" " :
      ELSE A=A+CR
1350 J=LEN(BL)-Z1 :
      IF J<Z1 THEN BL=A : RETURN :
      ELSE BL=LEFT$(BL,J)+A : RETURN
```

The EDTASM-Fixup subroutine changes an EDTASM file line into an ASCII file line, removing the leading coded statement number.

```
1400 J=ASC(BL) :
     IF J=LX THEN BL=".MEND" : RETURN :
      ELSE IF J=LQ THEN J=14 :
       ELSE IF J>LZ THEN J=Z7 :
        ELSE RETURN
1450 BL=MID$(BL,J) : J=ASC(BL) :
      IF J<>Z9 THEN RETURN :
       ELSE BL="  "+MID$(BL,Z2) : RETURN
```

Macro-Return into the main execution loop decrements the parameter stack pointer and frees the space held by the current macro parameters.

```
1550 FOR J=Z0 TO ZT :
      P(IP+J)="" :
     NEXTJ :
     IP=IP-ZT
```

The main execution loop gets the next line, substitutes variable values, and if the line is not a macro call, it outputs the line and loops. Note that ". ." is not treated as a macro call but as a request to generate a macro call. This allows the processor to be used to generate a control statement stream for later use.

```
1600 GOSUB 1200 : GOSUB 900 :
       IF LEFT$(BL,Z1)<>"." THEN GOSUB 1900 : GOTO 1600 :
       ELSE IF LEFT$(BL,Z2)=".." THEN
        BL=MID$(BL,Z2)  : GOSUB 1900 : GOTO 1600
```

This routine pushes the parameter stack and checks to see if the macro is built-in.

```
1650 IP=IP+ZT : GOSUB 500 : J=(INSTR(GS,P(IP))+Z4)/Z5
```

If the macro is built-in, a J-way branch is taken.

```
1700 ON J GOTO 4100 ,2150 ,2350 ,2350 ,1550 ,2500,
     2900 ,3200 ,3400 ,3500 ,3750 ,3800 ,3850,
     2700 ,2750 ,2800 ,4000 ,4250 ,4600 ,4450,
     4650 ,4750,1550
```

Since no branch was taken at 1700, this must be a user-defined macro. Search the name table GN for a match. If not found then ignore it.

```
1750 J=(INSTR(GN,P(IP))+Z4)/Z5 :
       IF J=Z0 THEN  GOTO 1550
```

These lines push the index of the macro's text lines onto the source stack so that they will be used as input.

```
1800 IS=IS+Z1 : S(IS)=IN(J) : SS(IS)=IN(J) : GOTO 1600
```

The Output subroutine sends the current line buffer BL to the output device selected by JO. It also traces the line by printing it on the screen if the option flag in JT is set. EDTASM format is used if the JA flag is set.

```
1900 IF (JO AND JI)=Z0 THEN
          PRINT "<";BL;">" : RETURN :
       ELSE IF (JT AND Z1)<>Z0 THEN
            PRINT "(";JI;")<";BL;">"
1950 IF (JA=JI) AND (CA=CR) THEN GOSUB 2050
2000 CA=RIGHT$(BL,Z1) : PRINT#JI,BL; : RETURN
```

Out-EDTASM formats a line with a line number for the NEWDOS EDTASM program.

```
2050 CN=RIGHT$("0000"+STR$(IA),Z5) : IA=IA+ZT :
     FOR I=Z1 TO Z5 :
      MID$(CN,I,Z1)=CHR$(KA OR ASC(MID$(CN,I,Z1))) :
     NEXT I :
     BL=CN+" "+BL : RETURN
```

.MAC saves user-assigned names and points each at the current top of the macro text buffer.

```
2150 J=Z1
2200 IF LEN(P(IP+J))>Z0 THEN
```

```
LN=LN+Z1 :
GN=GN+LEFT$("."+P(IP+J)+"      ",Z5) : J=J+Z1 :
IN(LN)=IG :
GOTO 2200
```

We now read all source up to a .MEND and place it in the macro text buffer without substitution for later use.

```
2250 GOSUB 1200 : G(IG)=BL : IG=IG+Z1 :
        IF LEFT$(BL,Z5)=".MEND" THEN 1550 :
        ELSE 2250
```

.MRET and .MEND close the current input source and pop the source stack. Input continues where it left off.

```
2350 J=S(IS) : IS=IS-Z1 :
      IF IS=-1 THEN
         GOSUB 4500 : CLOSE : PRINT"FINISHED" : STOP :
      ELSE IF J<Z0 THEN
           J=J AND &HF : CLOSE J : JF=JF-Z1
2400 GOTO 1550
```

The GOTO command lets AL = the label string.

```
2500 AL=P(IP+Z1)
```

Line 2550 searches the current source for a label that matches AL.

```
2550 GOSUB 1200 : IP=IP-ZT : GOSUB 900 : IP=IP+ZT :
        IF LEFT$(BL,Z5)=".MEND" THEN 2600 :
        ELSE IF LEFT$(BL,Z3)<>". : " THEN 2550 :
             ELSE GOSUB 500 :
                  IF P(IP+Z1)=AL THEN 1550 :
                  ELSE 2550
```

We are now at the end of the source. If the source is a macro, then restart at the top. If not, execute a .MEND.

```
2600 IF S(IS)<Z1 THEN 2350 :
        ELSE S(IS)=SS(IS) : GOTO 2550
```

Compare parameters 2 and 3 to .CIF = ,.CIF>, and .CIF<. If the result is true, then execute the .GOTO logic.

```
2700 IF P(IP+Z2)=P(IP+Z3) THEN 2500 : ELSE 1550
2750 IF P(IP+Z2)>P(IP+Z3) THEN 2500 : ELSE 1550
2800 IF P(IP+Z2)<P(IP+Z3) THEN 2500 : ELSE 1550
```

.SETC sets A equal to the proper substring of parameter 2.

```
2900 J=VAL(P(IP+Z3)) :
        IF J=Z0 THEN J=Z1
2950 K=VAL(P(IP+Z4)) :
        IF K=Z0 THEN K=LEN(P(IP+Z2))
3000 A=MID$(P(IP+Z2),J,K)
```

This instruction sets I to the index of the variable to be set in P.

```
3050 I=ASC(P(IP+Z1)) :
       IF I>58 THEN I=I-58 :
       ELSE I=IP+I-58
```

Line 3100 sets variable to the string A.

```
3100 P(I)=A : GOTO 1550
```

.INDX locates parameter 3 in parameter 2 and sets J to the location.

```
3200 J=INSTR(P(IP+Z2),P(IP+Z3))
```

J is converted to a string number in A. Then the program assigns its value to a proper variable.

```
3250 A=STR$(J) :
       IF LEFT$(A,Z1)=" " THEN A=MID$(A,Z2)
3300 GOTO 3050
```

.LEN sets J to the length of parameter 2, then assigns the value to a variable.

```
3400 J=LEN(P(IP+Z2)) : GOTO3250
```

.SETA evaluates the expressions in parameters 0 through 9 and puts the result in J. It then enters .LEN logic to assign the result to a variable.

```
3500 K=Z2 : J=VAL(P(IP+K))
3550 K=K+Z1 : A=LEFT$(P(IP+K),Z1) :
       IF A="" THEN 3250 :
       ELSE K=K+Z1 : I=VAL(P(IP+K))
3600 IF A="+" THEN J=J+I :
       ELSE IF A="-" THEN J=J-I :
       ELSE IF A="*" THEN J=J*I :
       ELSE IF A="/" THEN J=J/I :
       ELSE IF A="O" THEN J=J OR I :
       ELSE IF A="A" THEN J=J AND I :
       ELSE PRINT"** INVALID OPERATOR".BL
3650 GOTO 3550
```

.AIF, .AIF> and .AIF< arithmetically compare parameters 2 and 3. If the result is true, then enter the .GOTO logic.

```
3750 GOSUB 3900 :
       IF J=K THEN 2500 :
       ELSE 1550
3800 GOSUB 3900 :
       IF J>K THEN 2500 :
       ELSE 1550
3850 GOSUB 3900 :
       IF J<K THEN 2500 :
       ELSE 1550
```

Load J and K with parameters 2 and 3.

```
3900 J=VAL(P(IP+Z2)) : K=VAL(P(IP+Z3)) : RETURN
```

.SEQN sets J to the sequence number value and increments the counter. It then enters .LEN logic to set a variable to J's value.

```
4000 J=JS : JS=JS+Z1 : GOTO 3250
```

.IM pushes the new input source onto the source stack.

```
4100 A=P(IP+Z1) : IS=IS+Z1 :
       IFA="*KI" THEN S(IS)=Z0 : GOTO 1550
4150 S(IS)=JF OR &H8000 :
      OPEN "I",JF,A : JF=JF+Z1 : GOTO 1550
```

.OUTF sets the current output file # to 1.

```
4250 JI=Z1
```

If there is a null file name, then the process is finished. If not, CLOSE the old file if it is open.

```
4300 A=P(IP+Z1) :
       IF A="" THEN 1550 :
       ELSE IF (JO AND JI)<>Z0 THEN
         GOSUB 4500 : CLOSE JI : JO=((NOT JI)AND JO)AND Z3
```

OPEN the new file name. If there is no EDTASM option, then the process is finished.

```
4350 IF A="*DI" THEN  1550 :
       ELSE OPEN "O",JI,A : JO=JO OR JI :
       IF P(IP+Z2)<>"EDTASM" THEN JA=Z3 : GOTO 1550
```

Output an EDTASM file header.

```
4400 JA=JI : CA=CR : A=LEFT$(A+"       ",Z6) :
       PRINT#JI,CHR$(&HD3);A; : GOTO 1550
```

.OUT2 sets file #2 and enters .OUTF logic.

```
4450 JI=Z2 : GOTO 4300
```

If the EDTASM flag is on, then write a file trailer block.

```
4500 IF (JO AND JI)<>JA THEN RETURN :
       ELSE PRINT#JI,CHR$(&H1A); : RETURN
```

.INPT displays a message in parameter 2 and gets a new value for the variable name in parameter 1. Enter .SETC logic to assign the value.

```
4600 PRINT P(IP+Z2);LINEINPUT" : ";A : GOTO 3050
```

OPTN sets the option flag JT from parameter 1.

```
4650 JT=VAL(P(IP+Z1)) : GOTO 1550
```

.PSET resets the calling levels which the parameters set to those given on this macro invocation.

```
4750 FOR J=IP+Z1 TO IP+ZT :
       P(J-ZT)=P(J) :
      NEXTJ :
      GOTO 1550
```

```
    -----{   ***** INITIALIZE VARIABLES ***** }-----

4850 Z0=0 : J=Z0 : K=Z0 : I=Z0 : I=Z0 : IS=Z0 :
      DIM S(20) : DIM SS(20) : S(IS)=Z0 : JO=Z0 : JI=Z0
4900 Z1=1 : Z2=2 : Z3=3 : Z4=4 : Z5=5 : Z6=6 :
      Z7=7 : Z8=8 : Z9=9 : ZT=10
4950 BK=" " : BA="&" : ZA=ASC(BA) : BS=";" :
      BD="-" : CR=CHR$(13)
5000 DIM P(198)
5050 IP=39 : BL="" : DIM G(1000) : IG=Z1 :
      DIM IN(50) : GN="" : LN=Z0
5100 GS= ".IM  .MAC .MEND.MRET.:   .GOTO.SETC.INDX
     .LEN .SETA.AIF=.AIF>.AIF<.CIF=.CIF>.CIF<.SEQN
     .OUTF.INPT.OUT2.OPTN.PSET.REM "
5150 JF=Z3 : JT=Z0 : A="" : AL=""
5200 JA=Z3 : CA=BK : IA=ZT : KA=&HB0 : SS(IS)=Z0 :
      LX=&H1A : LQ=&HD3 : LZ=175 : JS=Z1
5250 RETURN
```

```
.OPTN 1
.REM THIS IS A SAMPLE PROGRAM TO TEST TRDOSIO/MAC
.REM NOW LETS GET THE DEFINITION FROM DISK
.IM TRDOSIO/MAC
.REM SETUP AN OUTPUT FILE
.OUTF TEST/OUT
.REM WRITE A HEADER INTO THE FILE
; THIS IS THE FIRST LINE IN TEST/OUT
; NOW LETS CALL THE OPEN MACRO
.OPEN FILE,BUFFER,LRECL
;THIS IS THE LAST LINE IN TEST
.REM CLOSE OUTPUT FILE
.OUTF *DI
.OPTN 0
.MEND.
```

Example 1

```
00
10 ; THIS IS THE FIRST LINE IN TEST/OUT
20 ; NOW LETS CALL THE OPEN MACRO
30          LD DE,FILE
40          LD HL,BUFFER
50          LD B,&3 ; RECORD LENGTH
60          CALL 4424H
70 ; THIS IS THE LAST LINE IN TEST
80 ;
```

Example 2

```
.REM TRDOS DISK I/O CALLS,
.REM      P1 IS DCB ADDR
.REM      P2 IS ADDR OF BUF/REC OR REC NUMBER
.REM      P3 IS OPEN RECORD LENGTH
.MAC INIT,OPEN,POSN,READ,WRIT,CLOS,KILL,VERF
    LD DE,&1
.GOTO &0
.: .POSN
    LD BC,&2
    CALL 4442H
.MRET
.: &0
```

```
.CIF= &0,+&2,+
    LD HL,&2
.GOTO &0
.: .INIT
    CALL 4420H
.MRET
.: .OPEN
    LD B,&3 ; RECORD LENGTH
    CALL 4424H
.MRET
.: .READ
    CALL 4436H
.MRET
.: .WRIT
    CALL 4439H
.MRET
.: .CLOS
    CALL 4428H
.MRET
.: .KILL
    CALL 442CH
.MRET
.: &0
 ; **** BAD CALL *****
.MEND
```

Example 3

A	String work area
AL	String save area for labels
BL	Input line buffer
RA	ENTER flag for EDTASM output
CN	Work area for EDTASM line number output
G	String array of macro source lines
GN	String of user-defined macro names
GS	String of built-in macro names
IA	EDTASM line number
IG	Last line used in macro source string array
IN	Array of starting-string indexes of macros stored in G
IP	Current index of source lines in G
IS	Index in S of current input source
JA	EDTASM output flag
JF	Next file number for an imbed
JI	Current output file
JO	Output-file-open flags
JS	.SEQN counter
JT	.OPTN flag
LN	Number of user-defined macro names
P	Variable and parameter stack
S	Array of input source flags (index or file number with high-order bit on)
SS	Stacks of starting line numbers of macros for .GOTO

Table 1. *BASIC variable usage*

Program Listing. *Micro-Mac*

Encyclopedia Loader

```
100 CLEAR 20000:
    DEFINT I - N:
    DEFINT Z,S:
    DEFSTR A - H:
    DEFSTR P
150 GOSUB 4800 :
    ' INITIALIZE VARIABLES
200 CLS :
    PRINT "MICRO-MACRO PROCESSOR  VERSION 1.0"
250 PRINT "          BY JOHN R. HIND"
300 PRINT CR;CR;CR;"ENTER MACRO LANGUAGE SOURCE"
350 PRINT "(.OUTF FN /.IM FN /ETC..)
400 GOTO 1600 :
    ' START EXECUTION
450 REM  ***  PARSE LINE BUFFER ***
500 J = INSTR (BL,BK):
    IF J > Z0
     THEN
      P(IP) = LEFT$( MID$(BL,Z1,J - Z1) + "      ",Z5):
      BL = MID$(BL,J + Z1, LEN(BL) - J - Z1):
      :
     ELSE
      P(IP) = LEFT$( LEFT$(BL, LEN(BL) - Z1) + "      ",Z5):
      BL = ""
550 FOR L = Z1 TO Z9:
     P(IP + L) = "":
     NEXT L:
    J = Z0
600 L = LEN(BL):
    IF L = Z0
     THEN
      RETURN :
      :
     ELSE
      A = LEFT$(BL,Z1):
      BL = MID$(BL,Z2):
      IF A = BK
       THEN
        600
650 J = J + Z1:
    IF A = ","
     THEN
      600 :
      :
     ELSE
      IF A < > "'" AND A < > ":"
       THEN
        750
700 K = INSTR (BL,A):
    P(IP + J) = LEFT$(BL,K - Z1):
    BL = MID$(BL,K + Z2):
    GOTO 600
750 K = INSTR (BL,","):
    IF K = Z0
     THEN
      K = L
800 P(IP + J) = A + LEFT$(BL,K - Z1):
    BL = MID$(BL,K + Z1):
    GOTO 600
850 REM  *** EXPAND TEXT LINE BUFFER ***
900 J = Z1:
    IF (JT AND Z1) = Z1
     THEN
      IF S(IS) < > Z0
       THEN
        PRINT "::";S(IS);"::";BL;"::"
950 J = INSTR (J,BL,BA):
      IF J = Z0
       THEN
        RETURN :
        :
       ELSE
        K = ASC( MID$(BL,J + Z1))
1000  IF K = ZA
       THEN
        BL = LEFT$(BL,J)  + MID$(BL,J + Z2):
        J = J + 1:
        GOTO 950
1050  IF K < 58
       THEN
        K = IP + K - 48:
        :
       ELSE
        IF K > 95
         THEN
          K = K - 90:
          :
         ELSE
          K = K - 58
1100  BL = LEFT$(BL,J - Z1) + P(K) + MID$(BL,J + Z2):
      GOTO 950
1150  REM  *** GET NEXT LINE ***
1200  J = S(IS):
      IF J > Z0
       THEN
        BL = G(J):
        S(IS) = J + Z1:

        RETURN
1250  J = J AND &HF:
      IF J = Z0
       THEN
        LINE INPUT "SOURCE INPUT : ";BL:
        :
       ELSE
        IF EOF (J)
         THEN
          BL = ".MEND":
          :
         ELSE
          LINE INPUT #J,BL:
          GOSUB 1400
1300  A = RIGHT$(BL,Z1):
      IF A = BS
       THEN
        A = CR:
        :
       ELSE
        IF A = BD
         THEN
          A = " ":
          :
         ELSE
          A = A + CR
1350  J = LEN(BL) - Z1:
      IF J < Z1
       THEN
        BL = A:
        RETURN :
        :
       ELSE
        BL = LEFT$(BL,J) + A:
        RETURN
1400  J = ASC(BL + " "):
      IF J = LX
       THEN
        BL = ".MEND":
        RETURN :
```

Program continued

```
        :
       ELSE
        IF J = LQ
         THEN
          J = 14:
          :
         ELSE
          IF J > LZ
           THEN
            J = Z7:
            :
           ELSE
            RETURN
1450 BL = MID$(BL,J):
     J = ASC(BL):
     IF J < > Z9
      THEN
       RETURN :
       :
      ELSE
       BL = "  " + MID$(BL,Z2):
       RETURN
1500 REM  *** EXECUTE LOOP ***
1550 FOR J = Z0 TO ZT:
      P(IP + J) = "":
      NEXT J:
     IP = IP - ZT
1600 GOSUB 1200 :
     GOSUB 900 :
     IF LEFT$(BL,Z1) < > "."
      THEN
       GOSUB 1900 :
       GOTO 1600 :
       :
      ELSE
       IF LEFT$(BL,Z2) = ".."
        THEN
         BL = MID$(BL,Z2):
         GOSUB 1900 :
         GOTO 1600
1650 IP = IP + ZT:
     GOSUB 500 :
     J = ( INSTR (GS,P(IP)) + Z4) / Z5
1700 ON J GOTO 4100 ,2150 ,2350 ,2350 ,1550 ,2500 ,2900 ,3200 ,3400 ,
     3500 ,3750 ,3800 ,3850 ,2700 ,2750 ,2800 ,4000 ,4250 ,4600 ,4450
      ,4650 ,4750,1550
1750 J = ( INSTR (GN,P(IP)) + Z4) / Z5:
     IF J = Z0
      THEN
       GOSUB 1900 :
       GOTO 1550
1800 IS = IS + Z1:
     S(IS) = IN(J):
     SS(IS) = IN(J):
     GOTO 1600
1850 REM  ***  OUTPUT A LINE BUFFER ***
1900 IF (JO AND JI) = Z0
      THEN
       PRINT "<";BL;">":
       RETURN :
       :
      ELSE
       IF (JT AND Z1) < > Z0
        THEN
         PRINT "(";JI;")<";BL;">"
1950 IF (JA = JI) AND (CA = CR)
      THEN
       GOSUB 2050
2000 CA = RIGHT$(BL,Z1):
     PRINT #JI,BL;:
     RETURN
```

```
2050 CN = RIGHT$("0000" + STR$(IA),Z5):
     IA = IA + ZT:
     FOR I = Z1 TO Z5:
      MID$(CN,I,Z1) = CHR$(KA OR ASC( MID$(CN,I,Z1))):
      NEXT I:
     BL = CN + " " + BL:
     RETURN
2100 REM  ***  .MAC ***
2150 J = Z1
2200 IF LEN(P(IP + J)) > Z0
      THEN
       LN = LN + Z1:
       GN = GN + LEFT$("." + P(IP + J) + "      ",Z5):
       J = J + Z1:
       IN(LN) = IG:
       GOTO 2200
2250 GOSUB 1200 :
     G(IG) = BL:
     IG = IG + Z1:
     IF LEFT$(BL,Z5) = ".MEND"
      THEN
       1550 :
       :
      ELSE
       2250
2300 REM  ***  .MEND & .RET  ***
2350 J = S(IS):
     IS = IS - Z1:
     IF IS = - 1
      THEN
       GOSUB 4500 :
       CLOSE :
       PRINT "FINISHED":
       CMD "S":
       :
      ELSE
       IF J < Z0
        THEN
         J = J AND &HF:
         CLOSE J:
         JF = JF - Z1
2400 GOTO 1550
2450 REM  ***  .SKIP ***
2500 AL = P(IP + Z1)
2550 GOSUB 1200 :
     IP = IP - ZT:
     GOSUB 900 :
     IP = IP + ZT:
     IF LEFT$(BL,Z5) = ".MEND"
      THEN
       2600 :
       :
      ELSE
       IF LEFT$(BL,Z3) < > ".: "
        THEN
         2550 :
         :
        ELSE
         GOSUB 500 :
         IF P(IP + Z1) = AL
          THEN
           1550 :
           :
          ELSE
           2550
2600 IF S(IS) < Z1
      THEN
       2350 :
       :
      ELSE
       S(IS) = SS(IS):
```

Program continued

```
      GOTO 2550
2650 REM  *** .CIF= / .CIF> / .CIF< ***
2700 IF P(IP + Z2) = P(IP + Z3)
      THEN
       2500 :
       :
      ELSE
       1550
2750 IF P(IP + Z2) > P(IP + Z3)
      THEN
       2500 :
       :
      ELSE
       1550
2800 IF P(IP + Z2) < P(IP + Z3)
      THEN
       2500 :
       :
      ELSE
       1550
2850 REM  *** .SETC ***
2900 J = VAL(P(IP + Z3)):
     IF J = Z0
      THEN
       J = Z1
2950 K = VAL(P(IP + Z4)):
     IF K = Z0
      THEN
       K = LEN(P(IP + Z2))
3000 A = MID$(P(IP + Z2),J,K)
3050 I = ASC(P(IP + Z1)):
     IF I > 58
      THEN
       I = I - 58:
       :
      ELSE
       I = IP + I - 58
3100 P(I) = A:
     GOTO 1550
3150 REM  *** .INDX ***
3200 J = INSTR (P(IP + Z2),P(IP + Z3))
3250 A = STR$(J):
     IF LEFT$(A,Z1) = " "
      THEN
       A = MID$(A,Z2)
3300 GOTO 3050
3350 REM  *** .LEN ***
3400 J = LEN(P(IP + Z2)):
     GOTO 3250
3450 REM  *** .SETA ***
3500 K = Z2:
     J = VAL(P(IP + K))
3550 K = K + Z1:
     A = LEFT$(P(IP + K),Z1):
     IF A = ""
      THEN
       3250 :
       :
      ELSE
       K = K + Z1:
       I = VAL(P(IP + K))
3600 IF A = "+"
      THEN
       J = J + I:
       :
      ELSE
       IF A = "-"
        THEN
         J = J - I:
         :
        ELSE
```

```
          IF A = "*"
           THEN
            J = J * I:
            :
           ELSE
            IF A = "/"
             THEN
              J = J / I:
              :
             ELSE
              IF A = "O"
               THEN
                J = J OR I:
                :
               ELSE
                IF A = "A"
                 THEN
                  J = J AND I:
                  :
                 ELSE
                  PRINT "** INVALID OPERATOR",BL
3650 GOTO 3550
3700 REM  *** .AIF= , .AIF> , .AIF<  ***
3750 GOSUB 3900 :
     IF J = K
      THEN
       2500 :
       :
      ELSE
       1550
3800 GOSUB 3900 :
     IF J > K
      THEN
       2500 :
       :
      ELSE
       1550
3850 GOSUB 3900 :
     IF J < K
      THEN
       2500 :
       :
      ELSE
       1550
3900 J = VAL(P(IP + Z2)):
     K = VAL(P(IP + Z3)):
     RETURN
3950 REM  *** .SEQN ***
4000 J = JS:
     JS = JS + Z1:
     GOTO 3250
4050 REM  *** .IM ***
4100 A = P(IP + Z1):
     IS = IS + Z1:
     IF A = "*KI"
      THEN
       S(IS) = Z0:
       PRINT BL:
       GOTO 1550
4150 S(IS) = JF OR &H8000:
     OPEN "I",JF,A:
     JF = JF + Z1:
     PRINT BL:
     GOTO 1550
4200 REM  *** .OUTF , .OUT2
4250 JI = Z1
4300 A = P(IP + Z1):
     IF A = ""
      THEN
       1550 :
       :
```

Program continued

```
      ELSE
       IF (JO AND JI) < > Z0
        THEN
         GOSUB 4500 :
         CLOSE JI:
         JO = (( NOT JI) AND JO) AND Z3
4350 IF A = "*DI"
      THEN
       1550 :
       :
      ELSE
       OPEN "O",JI,A:
       JO = JO OR JI:
       IF P(IP + Z2) < > "EDTASM"
        THEN
         JA = Z3:
         GOTO 1550
4400 JA = JI:
     CA = CR:
     A = LEFT$(A + "          ",Z6):
     PRINT #JI, CHR$(&HD3);A;:
     GOTO 1550
4450 JI = Z2:
     GOTO 4300
4500 IF (JO AND JI) < > JA
      THEN
       RETURN :
       :
      ELSE
       PRINT #JI, CHR$(&H1A);:
       RETURN
4550 REM  *** .INPT ***
4600 PRINT P(IP + Z2); LINE INPUT ": ";A:
     GOTO 3050
4650 JT = VAL(P(IP + Z1)):
     GOTO 1550
4700 REM  *** .PSET ***
4750 FOR J = IP + Z1 TO IP + ZT:
      P(J - ZT) = P(J):
      NEXT J:
     GOTO 1550
4800 REM  ***** INITIALIZE VARIABLES *****
4850 Z0 = 0:
     J = Z0:
     K = Z0:
     I = Z0:
     I = Z0:
     IS = Z0:
     DIM S(20):
     DIM SS(20):
     S(IS) = Z0:
     JO = Z0:
     JI = Z0
4900 Z1 = 1:
     Z2 = 2:
     Z3 = 3:
     Z4 = 4:
     Z5 = 5:
     Z6 = 6:
     Z7 = 7:
     Z8 = 8:
     Z9 = 9:
     ZT = 10
4950 BK = " ":
     BA = "&":
     ZA = ASC(BA):
     BS = ";":
     BD = "-":
     CR = CHR$(13)
5000 DIM P(198) :
        PARM STRINGS
5050 IP = 39:
     BL = "":
     DIM G(1000):
     IG = Z1:
     DIM IN(50):
     GN = "":
     LN = Z0
5100 GS = ".IM  .MAC .MEND.MRET.:   .GOTO.SETC.INDX.LEN .SETA.AIF=.AI
     F>.AIF<.CIF=.CIF>.CIF<.SEQN.OUTF.INPT.OUT2.OPTN.PSET.REM "
5150 JF = Z3:
     JT = Z0:
     A = "":
     AL = ""
5200 JA = Z3:
     CA = BK:
     IA = ZT:
     KA = &HB0:
     SS(IS) = Z0:
     LX = &H1A:
     LQ = &HD3:
     LZ = 175:
     JS = Z1
5250 RETURN
```

APPENDIX

Appendix A

Appendix B

APPENDIX

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P*.

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX

Glossary

A

ac input module—I/O rack module which converts various ac signals originating in user switches to the appropriate logic level for use within the processor.

ac output module—I/O rack module which converts the logic levels of the processor to a usable output signal to control a user's ac load.

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200-450 nanoseconds for TRS-80 RAM.

accumulator—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

active elements—any generators of voltage or current in an impedance network; also known as active components.

adaptor—a device for connecting parts that will not mate; a device designed to provide a compatible connection between systems or subsystems.

A/D converter—analog to digital converter. See D/A converter.

add with carry—a machine-language instruction in which one operand is added to another, along with a possible carry from the previous (lower-order) add.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications, generally on large-scale computers.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

alphanumerics—refer to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

Ampere—the unit of electric current in the meter-kilogram-second system of units; defined in terms of the force of attraction between two parallel current conductors; 1 coulomb/second.

Ampere-turn—a unit of magnetomotive force defined as the force of a closed loop of one turn with a current of one ampere flowing through the loop.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

analog input module—an I/O rack module which converts an analog signal from a user device to a digital signal which may be processed by the processor.

analog output module—an I/O rack module which converts a digital signal from the processor into an analog output signal for use by a user device.

AND—a Boolean logic function. Two operators are tested and, if both are true, the answer is true. Truth is indicated by a high bit, or 1 in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—a programming language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

arithmetic shift—a type of shift in which an operand is shifted right or left with the sign bit being extended (right shift) or maintained (left shift).

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

asynchronous—not related through repeating time patterns.

asynchronous shift register—a shift register which does not require a clock. Register segments are loaded and shifted only at data entry.

B

backup—1) refers to making copies of all software and data stored externally 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing in which many of the same types of jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud—1) a unit of data transmission speed equal to the number of code elements (bits) per second 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

binary digit—the two digits, zero and one, used in binary notation. Often shortened to bit.

binary point—the point, analagous to a decimal point, that separates the integer and fractional portions of a binary mixed number.

bipolar device—a device whose operation depends on the transport of holes and electrons, usually made of layers of silicon with differing electrical characteristics.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

borrow—one bit subtracted from the next higher bit position.

bps—bits per second.

breakdown—a large, abrupt rise in electric current due to decreased resistance in a semiconductor device caused by a small increase in voltage.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software or hardware.

bump contact—a large area contact used for alloying directly to the substrate of a chip for mounting or interconnecting purposes.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specially designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

card reader—a device for reading information from punched cards.

carrier—a steady signal that can be slightly modified (modulated) continuously. These modulations can be interpreted as data. In microcomputers the technique is used primarily in modern communications and tape input/output (I/O).

carry—a one bit added to the next higher bit position or to the carry flag.

carry flag—a bit in the microprocessor used to record the carry "off the end" as a result of a machine-language instruction.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode—in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

character—a single symbol that is represented inside the computer by a specific code.

charge—a basic property of elementary particles of matter. The charge, measured in coulombs, is the algebraic sum of the electric charge of its constituents.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for the block of data.

chip—the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit—a conductor or system of conductors through which an electric current may flow.

circuit card—a printed circuit board containing electronic components.

clear—to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

clobber—to destroy the contents of memory or a register.

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business-Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

Colossus—a British computer used to crack German Enigma codes during World War II.

common carrier—a communications transmission medium, such as the Direct Distance Dialing (DDD) network of the Bell System.

compiler—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

complementary functions—two driving point functions whose sum is a positive constant.

complementary metal oxide semiconductor—CMOS. A signal inverting device formed by the combination of a p channel with an n channel device usually connected in series across the power supply.

complementary transistors—two transistors of opposite conductivity (pnp and npn) in the same functional unit.

computer interface—a device designed for data communication between a central computer and another unit such as a programmable controller processor.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

conditional jump—a machine-language instruction that jumps if a specified flag (or flags) is set or reset.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

constant—a value that doesn't change.

control block—a storage area of a microprocessor containing the information required for control of a task, function, operation, or quantity of information.

coulomb—the unit of electric charge in SI units (International System of Units); the quantity of electric charge that passes any cross section of a conductor in one second when current is maintained constant at one ampere.

counter—in relay-panel hardware, an electro-mechanical device which can be wired and preset to control other devices according to the total cycles of one ON and OFF function. A counter is internal to the processor; i.e., it is controlled by a user-programmed instruction. A counter instruction has greater capability than any hardware counter.

CPU—central processing unit. The circuitry that actually performs the functions of the instruction set.

CRT—cathode ray tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to a read/write section of tape.

current—the net transfer or electric charge per unit of time by free electrons; 1 ampere = 1 coulomb/second.

current mode logic—CML. Integrated circuit logic in which transistors are paralleled so as to eliminate current hogging.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

daisy chain—a bus line which interconnects devices for serial operation.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

data base—refers to a series of programs each having a different function, but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

data link—equipment, especially transmission cables and interface modules, which permits the transmission of information.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

die bond—a process in which chips are joined to a substrate.

differential discriminator—a circuit that passes only pulses whose amplitudes are between two predetermined values, neither of which are zero.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

diode transistor logic—a circuit that uses diodes, transistors, and resistors to provide logic functions.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—DOS. The system software that manipulates the data to be sent to the disk controller.

displacement—a signed value in machine language used in defining a memory address.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

DMA—direct memory access. A process where the CPU is disabled or bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

domain—a region in a solid within which elementary atomic, molecular, magnetic, or electric moments are uniformly arrayed.

doping—the addition of impurities to a semiconductor to achieve a desired characteristic.

dot-matrix printer—instead of each letter having a separate type head (like a typewriter), a single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to manufacture.

double-dabble—a method of converting from binary to decimal representation by doubling the leftmost bit, adding the next bit, and continuing until the rightmost bit has been processed.

downtime—the time when a system is not available for production due to required maintenance.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrical Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editors.

electron—a stable elementary particle with a negative electric charge of about -1.602×10^{-19} coulomb.

emitter-coupled logic—a form of current mode logic in which the emitters of two transistors are connected to a single current-carrying resistor in a way that only one transistor conducts at a time.

enhancement mode—operation of a field effect transistor in which no current flows when zero gate voltage is applied, and increasing the gate voltage increases the current.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power of two of a floating-point number.

F

feedback—the signal or data fed back to the programmable controller from a controlled machine or process to denote its response to the command signal.

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

Fibonacci series—the sequence of number 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . in which each term is computed by addition of the two previous terms.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

filter—electrical device used to suppress undesirable electrical noise.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flip chip—a tiny semiconductor die having terminations all on one side in the form of solder pads or bump contacts; after the surface of the chip has been passivated or otherwise treated, it is flipped over for attaching to a matching substrate.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size of number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

half duplex—data can flow in both directions, but not simultaneously. See duplex.

Hall effect—the development of a transverse electric field in a current-carrying conductor placed in a magnetic field; ordinarily the conductor is positioned so that the magnetic field is perpendicular to the direction of current flow and the electric field is perpendicular to both.

Hall generator—a generator using the Hall effect to give an output voltage proportional to magnetic field strength.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hangup—the computer has ceased processing inexplicably.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hard magnetic—a term describing a metal having a high coercive force which gives a high magnetic hysteresis; usually a permanent magnetic material.

hard wired—having a fixed wired program or control system built in by the manufacturer and not subject to change by programming.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexa-dabble—conversion from hexadecimal to decimal by multiplying each hex digit by sixteen and adding the next hex digit until the last (rightmost) hex digit has been reached.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

hole—a mobile vacancy having an energy state near the top of the energy band of a solid; behaves as though it were a positively charged particle.

host computer—the primary computer in a multi-computer or terminal hookup.

human engineering—usually refers to designing hardware and software with ease of use in mind.

hysteresis—an oscillator effect wherein a given value of an operating parameter may result in multiple values of output power or frequency.

I

IC—integrated circuit.

immediate—addressing mode in which the address of the information that an operation is supposed to act upon immediately follows the operation code.

inclusive OR—a bit-by-bit logical operation which produces a one-bit result if one or the other operand bits, or both is a one.

increment—to increase, usually by one. See decrement.

indexed—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one certain prescribed operation.

insulator—a nonconducting material used for supporting or separating conductors to prevent undesired current flow to other objects.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

integrated injection logic—I^2L. Integrated circuit logic which uses a simple and compact bipolar transistor gate structure which makes possible large scale integration on silicon for logic arrays and other analog and digital applications.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compiler.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

I/O module—the printed circuit board that is the termination for field wiring of I/O devices.

I/O rack—a chassis which contains I/O modules.

I/O scan—the time required for the programmable controller processor to monitor all inputs and control all outputs. The I/O scan repeats continuously.

isolated I/O module—a module which has each input or output electrically isolated from every other input or output on that module. That is to say, each input or output has a separate return wire.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

Josephson effect—the tunneling of electron pairs through a thin insulating barrier between two superconducting materials.

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

Karnaugh map—a truth table that shows a geometrical pattern of functional relationships for gating configurations; with this map, essential gating requirements can be recognized in their simplest form.

L

ladder diagrams—an industry standard for representing control logic relay systems.

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

latching relay—a relay with 2 separate coils, one of which must be energized to change the state of the relay; it will remain in either state without power.

leakage current—in general, the undesirable flow of current through or over the surface of an insulating material or insulator; the alternating current that passes through a rectifier without being rectified.

leakage flux—magnetic lines of force that go beyond their intended space.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that converts electric energy efficiently into spontaneous and noncoherent electromagnetic radiation at visible and near infrared wavelengths of electroluminescence at a forward biased pn junction.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line driver—an integrated circuit specifically designed to transmit digital information over long lines—that is, extended distances.

line printer—a high-speed printing device that prints an entire line at one time.

linear circuit—a network in which the parameters of resistance, inductance, and capacitance are constant with respect to current or voltage, and in which the voltage or current of sources is independent of or directly proportional to the outputs.

linearity—the relationship that exists between two quantities when a change in one of them produces a direct proportional change in the other.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value, the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

lsb—see least significant bit.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

magnetoresistor—magnetic field controlled variable resistor.

magnitude—the absolute value, independent of direction.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

memory—the hardware that stores data for use by the CPU. Each piece of data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

memory module—a processor module consisting of memory storage and capable of storing a finite number of words (e.g., 4096 words in a 4K memory module). Storage capacity is usually rounded off and abbreviated with K representing each 1024 words.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—μs. One millionth of a second: 1×10^{-6} or 0.000001 second.

millisecond—μs. One thousandth of a second: 10^{-3} or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

monitor—1) a CRT 2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiple-precision numbers—multiple-byte numbers that allow extended precision.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, each requiring four data lines, can all be written to with only one group of four data lines. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplicand register—the register used to hold the multiplicand in a machine-language multiply.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

n-channel—a conduction channel formed by electrons in an n-type semiconductor, as in an n-type field-effect transistor.

negation—changing a negative value to a positive value, or vice versa. Taking the two's complement by changing all ones to zeros, all zeros to ones, and adding one.

nesting—putting one loop inside another. Some computers limit the number of loops that can be nested.

network—a collection of electric elements, such as resistors, coils, capacitors, and sources of energy, connected together to form several interrelated circuits. A collection of computer terminals interconnected to a host CPU.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

normalization—converting data to a standard format for processing. In floating-point format, converting a number so that a significant bit (or hex digit) is the first bit (or four bits) of the fraction.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

NPN transistor—a junction transistor having a p-type base between an n-type emitter and an n-type collector; the emitter should then be negative with respect to the base and the collector should be positive with respect to the base.

n-type semiconductor—an extrinsic semiconductor in which the conduction electron density exceeds the hole density.

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0–7.

octal-dabble—conversion of an octal number to decimal by multiplying by eight and adding the next octal digit, continuing until the last (rightmost) digit has been added.

OEM—Original Equipment Manufacturer.

off-line—describes equipment or devices which are not connected to the communications line.

offset value—a value that can be added to an address. Most addressing modes allow an offset value.

off-the-shelf—a term referring to software. A generalized program that can be used by a greater number of computer owners, so that it can be mass produced and bought off-the-shelf.

Ohm—the unit of resistance of a conductor such that a constant current of one ampere in it produces a voltage of one volt between its ends.

Ohm's law—a fundamental rule of electricity; states that the current in an electric circuit is inversely proportional to the resistance of the circuit and is directly proportional to the electromotive force in the circuit. In its strictest sense, Ohm's law applies only to linear constant-current circuits.

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

on-line operation—operations where the programmable controller is directly controlling the machine or process.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

oscillation—any effect that varies periodically back and forth between two values, as in the amplitude of an alternating current.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

overflow flag—a bit in the microprocessor used to record an overflow condition for machine-language operation.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

padding—filling bit positions to the left with zeros to make a total of eight or sixteen bits.

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word depends on the computer. Most micros are eight-bit word machines. Many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously.

parallel circuit—an electric circuit in which the elements, branches (having elements in series), or components are connected between two points, with one of the two ends of each component connected to each point.

parallel operation—type of information transfer whereby all digits of a word are handled simultaneously.

parallel output—simultaneous availability of two or more bits, channels, or digits.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even, the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check—a check that tests whether the number of 1s in an array of binary digits is odd or even.

partial product—the intermediate results of a multiply. At the end, the partial product becomes the whole product.

partial product register—the register used to hold the partial results of a machine-language multiply.

passivation—growth of an oxide layer on the surface of a semiconductor to provide electrical stability by isolating the transistor surface from electrical and chemical conditions in the environment; this reduces reverse-current leakage, increases breakdown voltage, and raises power dissipation rating.

passive element—an element of an electric circuit that is not the source of energy, such as a resistor, inductor, or capacitor.

PC—see programmable controller.

PC board—see printed circuit board.

p-channel—a conduction channel formed by holes in a p-type semiconductor, as in a p-type field effect transistor.

peripheral devices—a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permeability—a factor, characteristic of a material, that is proportional to the magnetic induction produced in a material divided by the magnetic field strength given by the equation:

$$m = \frac{\text{magnetic induction (gauss)}}{\text{magnetizing field (oersteds)}}$$

permutation—arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1—an acronym for programming language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter—a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

P-N junction—a region of transition between p-type emitter and n-type semiconducting regions in a semiconductor device.

PNP transistor—a junction type transistor having an n-type base between a p-type emitter and a p-type collector.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

print buffer—a portion of memory dedicated to holding the string of characters to be printed.

printed circuit board—a piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

program panel—a device for inserting, monitoring, and editing a program in a programmable controller.

program scan—the time required for the programmable controller processor to execute all instructions in the program once. The program scan repeats continuously. The program monitors inputs and controls outputs through the input and output image tables.

programmable controller—PC. A solid state control system which has a user-programmable memory for storage of instructions to implement specific functions such as I/O control logic, timing, counting, arithmetic, and data manipulation. A PC consists of the central processor, input/output interface, memory, and programming device which typically uses relay-equivalent symbols. The PC is purposely designed as an industrial control system which can perform functions equivalent to a relay panel or a wired solid state logic control system.

PROM—Programmable Read Only Memory. A memory device that is written to once and from then on acts like a ROM.

protocol—a defined means of establishing criteria for receiving and transmitting data through communication channels.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

p-type semiconductor—an extrinsic semiconductor in which the hole density exceeds the conduction electron density.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of divident remaining after a divide has been completed.

residue—the amount of dividend remaining, part way through a divide.

resistor-transistor logic—RTL. One of the simplest logic circuits, having several resistors, a transistor, and a diode.

resolution—a measure of the smallest possible increment of change in the variable output of a device.

restoring divide—a divide in which the divisor is restored if the divide "does not go" for any iteration. A common microcomputer divide technique.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

rotate—a type of shift in which data is recirculated right or left back into the operand from the opposite end.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

rung—a grouping of PC instructions which controls one output. This is represented as one section of a logic ladder diagram.

S

scaled up—referring to a number which has been multiplied by a scale factor for processing.

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scan time—the time necessary to completely execute the entire programmable controller program one time.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

self-diagnostic—the hardware and firmware within a controller which allows it to continuously monitor its own status and indicate any fault which might occur within.

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and are called semiconductors.

semiconductor device—an electronic device in which the characteristic distinguishing electronic conduction takes place within a semiconductor.

sensor—a sensing element, a device which senses either the absolute value or the change in a physical quantity, and converts that change into a useful signal for an information-gathering system.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. contrast with parallel.

serial operation—type of information transfer within a programmable controller whereby the bits are handled sequentially rather than simultaneously, as they are in parallel operation. Serial operation is slower than parallel operation for equivalent clock rates. However, only one channel is required for serial operation.

series circuit—a circuit in which all parts are connected end to end to provide a single path for current.

shift and add—a multiply method in which the multiply is achieved by shifting of and addition of the multiplicand.

shift register—a program, entered by the user into the memory of a programmable controller, in which the information data (usually single bits) is shifted one or more positions on a continual basis. There are two types of shift registers: asynchronous and synchronous.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (–) and 0 is positive (+).

sign extension—extending the sign bit of a two's complement number to the left by a duplication.

sign flag—a bit in the microprocessor used to record the sign of the result of a machine-language operation.

sign-magnitude—a nonstandard way of representing positive and negative numbers in microcomputers.

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

silicon controlled rectifier—SCR. A semiconductor rectifier that can be controlled; it is a pnpn four-layer semiconductor device that normally acts as an open circuit, but switches rapidly to a conducting state when an appropriate gate signal is applied to the gate terminal.

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

sink—a device that drains energy off a system; a device that switches a load to an absorbing material, such as a ground.

software—refers to the programs that can be run on a computer.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

SOS—silicon on sapphire. A semiconductor manufacturing technology in which metal oxide semiconductor devices are constructed in a thin single-crystal silicon film grown on an electrically insulating synthetic sapphire substrate.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

special purpose logic—proprietary features of a programmable controller which allow it to perform logic not normally found in relay ladder logic.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

start-up—the time between equipment installation and the full operation of the system.

state—the logic 0 or 1 condition in programmable controller memory or at a circuit's input or output.

status register—the register that contains the status flags set and tested by the CPU operations.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

storage—see memory.

strip printer—a peripheral device used with a programmable controller to provide a hard copy of process numbers, status, and functions.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

substrate—the physical material on which a microcircuit is fabricated; used primarily for mechanical support and insulating purposes; however, semiconductor and ferrite substrates may also provide useful electric functions.

subtract with carry—a machine-language instruction in which one operand is subtracted from another, along with a possible borrow from the next lower byte.

subtrahend—the number that is subtracted from the minuend.

successive addition—a multiplication method in which the multiplicand is added a number of times equal to the multiplier to find the product.

surge—a transient variation in the current and/or potential at a point in the circuit.

synchronous shift register—shift register which uses a clock for timing of a system operation and where only one state change per clock pulse occurs.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

tape reader—a unit which is capable of sensing data from punched tape.

Teletype™—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

termination—1) the load connected to the output end of a transmission line 2) the provisions for ending a transmission line and connecting to a bus bar or other terminating device.

text editor—see word processor.

thumbwheel switch—a rotating numeric switch used to input numeric information to a controller.

timer—in relay-panel hardware, an electromechanical device which can be wired and preset to control the operating interval of other devices. In the programmable controller a timer is internal to the processor, which is to say it is controlled by a user-programmed instruction. A timer instruction has greater capability than any hardware timer. Therefore, programmable controller applications do not require hardware timers.

time sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

transducer—a device used to convert physical parameters, such as temperature, pressure, and weight into electrical signals.

translator package—a computer program which allows a user program (in binary) to be converted into a usable form for computer manipulation.

transistor—an active component of an electronic circuit consisting of a small block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1 and 0 volts is low or 0; 5V = 1, 0V = 0).

Triac^{TM}—a General Electric trademark for a gate controlled semiconductor switch designed for alternating current power control; with phase control of the gate signal, load current can be varied over a range from 5 percent to 95 percent of full power.

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

volt—the unit of potential difference or electromotive force in the meter-kilogram-second system, equal to the potential difference between two points for which 1 coulomb of electricity will do 1 joule of work in going from one point to the other.

voltage—potential difference or electromotive force capable of producing a current; measured in volts.

voltage drop—the voltage developed across a component or a conductor by the flow of current through the resistance or impedance of the component or conductor.

von Neumann, John (1903–1957)—Mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. If the CPU works with 8 bits, then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

INDEX

INDEX

INDEX COMPILED BY NAN MCCARTHY

A. How did you learn about the Encyclopedia?

☐ Ads in magazines	☐ Gift	☐ Word of mouth
☐ Book review	☐ Saw it in a store	☐ Other

B. What Model(s) TRS-80 do you own?

☐ Model I Level I	☐ Model II	☐ Model III Level I
☐ Model I Level II	☐ Color computer	☐ Model III Level II

C. How much memory capacity does your computer have?

☐ 4 K ☐ 16 K ☐ 32 K ☐ 48 K

D. What peripheral(s) do you own?

☐ Disk drive	☐ Joy stick	☐ Sound mod
☐ Expansion interface	☐ Modem	☐ Stringy floppy
☐ Fixed disk	☐ Printer	Other ________
		________

E. If you were to buy an additional microcomputer, what system would you choose?

☐ Apple	☐ Hewlett Packard	☐ TRS-80
☐ Atari	☐ IBM	☐ Xerox
☐ Heath	☐ Pet	Other ________
		________

F. What language(s) do you use?

☐ Assembler	☐ FORTRAN	☐ PASCAL
☐ BASIC	☐ PILOT	Other ________
☐ COBOL	☐ FORTH	________

G. What language(s) would you like to learn?

☐ Assembler	☐ FORTRAN	☐ PASCAL
☐ BASIC	☐ PILOT	Other ________
☐ COBOL	☐ FORTH	________

H. What is your primary use of the TRS-80?

☐ Self-employed business	☐ Home applications	☐ Use at work
☐ Education/School	☐ Personal finance	Other ________
☐ Family education	☐ Scientific	________

I. The articles in the Encyclopedia are:

☐ Too simple ☐ Too complex ☐ Just right

J. Do you feel that the articles and programs have enough documentation? ☐ Yes ☐ No

K. Has the Encyclopedia helped you to improve your programming skills? ☐ Yes ☐ No

L. Please evaluate each section of the Encyclopedia in terms of its interest and value to you.

	Little/No Interest	Moderate Interest	Most Interest
Business	☐	☐	☐
Education	☐	☐	☐
Games	☐	☐	☐
Graphics	☐	☐	☐
Hardware	☐	☐	☐
Home Applications	☐	☐	☐
Interface	☐	☐	☐
Language	☐	☐	☐
Tutorial	☐	☐	☐
Utility	☐	☐	☐

M. Does the overall quality of the Encyclopedia match your expectations?
Please indicate on a scale of 1 to 4.

☐ 1. disappointed ☐ 2. mediocre ☐ 3. satisfied ☐ 4. pleased

ENCYCLOPEDIA SURVEY

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 81 PETERBOROUGH NH 03458

POSTAGE WILL BE PAID BY ADDRESSEE

WAYNE GREEN BOOKS
Summer St.
Peterborough NH 03458

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

ISBN 0-88006-036-0